UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jan Hanzel

UČENJE STRATEGIJE INTERAKTIVNEGA UČENJA Z GENETSKIM ALGORITMOM

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: doc. dr. Danijel Skočaj

Ljubljana, 2014

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani **Jan Hanzel**, z vpisno številko **63050040**, sem avtor diplomskega dela z naslovom:

Učenje strategije interaktivnega učenja z genetskim algoritmom.

S svojim podpisom izjavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Danijela Skočaja,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) in ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 1.10.2014 Podpis avtorja:

Zahvala

Za vso pomoč in potrpljenje pri izdelavi diplomskega dela se iskreno zahvaljujem mentorju doc. dr. Danijelu Skočaju.

Velika hvala moji družini, ki me je med študijem podpirala in mi ga sploh omogočila – hvala mama Ana, oče Borut in sestra Gaja.

Kazalo

Povzetek

Cilj diplomskega dela je bil razviti algoritem za učenje najboljše učne strategije v primeru interaktivnega učenja med robotom in človekom. Predstavili smo definicijo in formalizacijo učne strategije, ki določa obnašanje učenca in učitelja v interaktivnem učnem procesu. Predstavili smo tudi genetski algoritem, s katerim smo reševali naš optimizacijski problem. Vektorje, s katerimi so predstavljene učne strategije, smo vrednotili, križali in mutirali ter tako poskušali priti z vsako iteracijo do vedno boljše učne strategije. Vektorje smo vrednotili z vrednostjo mere prepoznavanja, ki nam pove, kako je bil uspešen algoritem pri prepoznavanju določenega koncepta po učenju z neko učno strategijo, in ceno tutorstva, ki nam pove, koliko je v učnem procesu sodeloval tutor. Rezultati so pokazali, da skozi iteracije genetskega algoritma narašča kakovost učnih strategij. To se odraža na povečani uspešnosti vrednosti mere prepoznavanja in zmanjšanju cene tutorstva.

Ključne besede:

Interaktivno učenje, učne strategije, genetski algoritem.

Abstract

The main goal of this thesis was to develop an algoritem for learning the best strategy in the case of interactive learning between a human and a robot. We presented the definition and formalization of a learning strategy. A learning strategy specifies the behaviour of a student and a teacher in a interactive learning process. We also presented a genetic algorithm to resolve our optimisation problem. We tryed to inpruve vectors which are used to present learning strategies. The vectors were evaluated, crossed and mutated each iteration. For evaluation of the vectors we used recognition score which measures how successful was the algorithm at recognizing a specific concept after learning it by using a specific strategy. We also used tutoring cost to measure the tutor involvment in the learning process. The results show that the quality of the strategies increases each iteration. We notice the inprovement as an increase in the recognition score and decrease of tutoring cost.

Key words:

Interactive learning, learning strategies, genetic algorithm.

1 UVOD

Ljudje se od svojega rojstva naprej učimo. V prvih letih življenja se naši možgani učijo prek lastnih opažanj, do katerih pridemo s pomočjo čutnih zaznav. Govora in vedenjskih vzorcev se naučimo skozi opazovanje in oponašanja ljudi okoli nas, ki opazujejo naš napredek in nas sproti popravljajo, če se česa nismo naučili pravilno, npr. izgovorjave določenih glasov. Ta glas nam ponavljajo, dokler ga tudi sami ne znamo pravilno oblikovati. Zdi se, da je interaktivno učenje, pri katerem se učenec uči prek lastnih opažanj in interakcije z učiteljem, za ljudi naravno.

Pomembnost takega učenja za dojenčke je predstavil Roy Pea [1]. Da so stiki z ostalimi ljudmi ključni za mentalni razvoj posameznika, je predlagal Lev Vygotsky [2]. Tak način učenja nam omogoča, da svoje znanje pridobivamo, nadgrajujemo in popravljamo v različnih okoljih skozi celo življenje.

V prihodnosti pričakujemo, da bodo del našega okolja robotski pomočniki, ki nam bodo pomagali pri vsakdanjem življenju. Da bi ti dobro delovali v človeškem okolju, od njih pričakujemo določene lastnosti, ki so ljudem sorodne. Poleg določenih fizičnih podobnosti ljudi, ki bi jim pomagale pri delovanju v tem okolju, se bodo morali roboti predvsem znajti v človeškem okolju, kot se znajdejo ljudje sami. Del sposobnosti prilagajanja pa leži ravno v učenju. Pomoč tutorja v primerih strojnega učenja predlaga Weng [3], Thomaz [4] pa predstavi medsebojni vpliv človeka in stroja za reševanje problemov strojnega učenja. Interaktivno učenje z lastnostmi, ki smo jih prej pripisali ljudem, bi bilo tako za robote, ki delujejo v človeškem okolju, potrebno orodje za njihovo uspešno delovanje.

Primer učenja robota v vsakdanjem stiku s človekom bi bilo učenje prepoznavanja nekega novega predmeta. Recimo, da nekdo robota poskuša naučiti prepoznati skodelico. Robotu pokaže skodelico, ta pa si na podlagi lastnega opazovanja ustvari model skodelice. Model mu bo služil za kasnejše prepoznavanje tega predmeta. Nato človek ukaže robotu, naj mu prinese skodelico. Lahko se zgodi, da bo prinesen predmet res skodelica, lahko pa bo npr. kozarec. Človek mu lahko pove, da to ni skodelica, lahko pa mu tudi spet pokaže neko skodelico. Na podlagi teh novih povratnih informacij robot popravi svoj model o tem predmetu in morda bo naslednjič uspešnejši pri prepoznavanju. Na tak način bi bili lahko roboti morda dovolj prilagodljivi, da bi skozi svoje delovanje pridobivali novo znanje in posodabljali obstoječega.

Pri interaktivnem učenju nista pomembna samo učenec in učitelj, temveč je pomembno tudi to, kako poteka sama interakcija med njima. Pomembno je, da je ta interakcija učinkovita, neučinkovita pa lahko vodi k slabšim rezultatom učenja. V tem diplomskem delu bomo predstavili interakcijo kot učno strategijo, ki nam natančno podaja, kakšne akcije lahko izvedeta učenec in tutor ter kdaj jih izvedeta med učenjem. Učenje učinkovite učne strategije

pomeni iskanje takega obnašanja učenca in tutorja v interaktivnem učnem procesu, ki pripelje do čim boljših učnih rezultatov učenca ob čim manjšem delu tutorja. Gre za optimizacijski problem. Poznamo različne optimizacijske metode, npr. dinamično programiranje, roj delavcev, genetski algoritmi itd. Zaradi načina formalizacije učnih strategij, ki je predstavljena v tem delu, sta za učenje učnih strategij primerni uporaba in implementacija genetskega algoritma. To je razlog, da smo genetski algoritem uporabili za reševanje tega problema.

V diplomskem delu smo se poskušali naučiti najboljšo učno strategijo. Predpostavili smo scenarij robota, ki se poskuša nekaj naučiti od človeka. Za učenje robota smo uporabili učni algoritem, za učitelja pa program tutorja, ki simulira človeški ekspert. Učni algoritem se je učil prepoznavanja barv. Učenec je lahko pridobival konceptualno znanje s pomočjo lastne interakcije s podanimi primeri in interakcije s tutorjem. Določili smo možne akcije učenca in tutorja. Učna strategija določa, katere akcije bosta uporabila učenec in tutor med potekom učnega procesa. Z genetskim algoritmom smo se poskušali naučiti zaporedja možnih akcij tutorja in učenca v učnem procesu, ki pripada najboljši učni strategiji. V ta namen smo razvili tudi način vrednotenja učne strategije.

Diplomsko delo je sestavljeno iz šestih poglavij. Uvodu sledi predstavitev interaktivnega učenja, formalizacija učnih strategij in nekaj njenih primerov. Tretje poglavje vsebuje splošno predstavitev genetskega algoritma in idejo o tem, kako smo ga uporabili za iskanje čimbolj učinkovitih učnih strategij. Četrto poglavje prikazuje programsko implementacijo genetskega algoritma za reševanje našega problema. Predzadnje poglavje je namenjeno rezultatom in metodam vrednotenja učnih strategij, zadnje poglavje pa je namenjeno zaključku.

2 INTERAKTIVNO UČENJE

Kot smo povedali že v uvodu, je interaktivno učenje takšno učenje, pri katerem učenec komunicira s svojim okoljem. Če komunicira z nekim učiteljem, pomeni, da mu podaja neko povratno informacijo o poteku učenja, na to informacijo pa se učitelj odzove in pomaga učencu. Interaktivno učenje je tako primerno za nek dolg proces učenja in služi ne samo temu, da se pridobiva popolnoma novo znanje o nečem, ampak da se že obstoječe znanje nadgrajuje.

2.1 Zahteve učnega algoritma

Učenje v računalniških sistemih poteka z učnim algoritmom, s katerim se ustvari model koncepta, ki smo se ga naučili. Od učnega algoritma, ki se uporabi v interaktivnem učenju, zahtevamo določene sposobnosti, ki so nujno potrebne za uporabo učnih strategij.

Ljudje smo sposobni svoje znanje o nečem izboljšati. Enako lahko zahtevamo od učnega algoritma. Vsakič, ko pride do novih podatkov o že naučenem konceptu, bodisi prek lastnih opazovanj ali s pomočjo tutorja, mora algoritem omogočiti posodobitev tega koncepta skladno z novim dognanjem.

Pri interaktivnem učenju s tutorjem lahko ta učenca opozori, če opazi, da se je nečesa narobe naučil. Učni algoritem učenca mora imeti sposobnost, da svoje napačno znanje popravi. Do napačnega znanja učenca pride predvsem, kadar se uči samostojno. Pri napačni razlagi učnega primera se predstavitev modela koncepta v učnem algoritmu popači, kar lahko sčasoma privede do vedno večje popačitve modela, ki ni več konsistenten s konceptom, ki ga predstavlja [5].

Zadnja zahteva za učni algoritem je tudi povezana z njegovo interakcijo, ki jo ima s tutorjem. V določenih scenarijih mora učenec prevzeti pobudo in prositi tutorja za pomoč. Da lahko to stori, mora biti učni algoritem sposoben ocene svojega lastnega znanja o nekem konceptu. Če oceni, da sam ni sposoben zanesljivo prepoznati nekega učnega primera, potem mora prositi za pomoč [5].

2.2 Učne strategije

Za definiranje učne strategije bomo vzeli sledečo definicijo:

»Učno strategijo definiramo kot običajno strategijo tutorja in robota, ki natančno opredeli obnašanje robota in tutorja v nepretrganem učnem procesu. Natančno opredeli, kdaj učenec posodobi znanje samodejno in kako ga posodobi ter kdaj in kako tutor in robot komunicirata z namenom, da razširita učenčevo znanje.« (Skočaj et al., 2009) [5].

2.2.1 Osnovne učne strategije

Iz prejšnje definicije lahko izpeljemo nekaj osnovnih učnih strategij, ki nam bodo olajšale kasnejše razumevanje oziroma reševanje problema najboljših strategij. Različne učne strategije se med seboj razlikujejo predvsem po različnih nivojih interakcije. Pri prvih dveh je učni proces v večji meri odvisen od tutorja, pri zadnjih pa se poveča vloga učenca. Štiri osnovne učne strategije so [5]:

- **popolnoma usmerjena (angl. *tutor-driven*)**: učenčeva vloga je minimalna, saj je interakcija med njim in tutorjem enosmerna ter v celoti vodena s strani slednjega. Vse informacije, potrebne za učenje učenca, poda tutor. Učenec dobi opis opazovanja in se iz tega uči;
- **nadzorovana (angl. *tutor-supervised*)**: učenčeva vloga se v tem načinu poveča, saj se uči iz lastnih opazovanj. Nad njegovim učenjem bedi tutor, ki ob napačni interpretaciji priskoči na pomoč. Učencu poda informacije, s katerimi lahko ta svoje znanje o opazovanju nato bodisi popravi bodisi posodobi;
- **asistirana (angl. *tutor-assisted*)**: vloga učenca je sedaj večja kot vloga tutorja. Učenje učenca poteka na enak način kot pri »tutor-supervised«. Razlika je v tem, da tutor ne bdi nad učenjem učenca, temveč priskoči na pomoč le takrat, ko zanjo zaprosi učenec;
- **popolnoma samostojna (angl. *tutor-unassisted*)**: tutorjeva vloga je zanemarljiva, saj ta ne prisostvuje pri učnem procesu. Vse učenje opravi učenec sam, brez kakršne koli pomoči.

Vidimo lahko, da v zadnjem primeru tutor ni potreben. Vse učenje pade na ramena robota, pri ostalih treh pa je tutor prisoten, ampak na različnih nivojih. Vloga učenca je še vedno zelo pomembna pri vodenju interakcije s tutorjem v predzadnjem primeru. Tukaj tutor pomaga le, če je pozvan. V drugem primeru učitelj prevzame nalogo nadzornika in tako razbremeni robota po odločanju, kdaj vprašati za pomoč. V prvi strategiji pa je učenec popolnoma odvisen od tutorja, saj mu ta podaja vse informacije, potrebne za učenje. Možne so tudi druge učne strategije, ki bodo predmet našega iskanja učnih strategij. Tukaj smo navedli le štiri, ki jih bomo potrebovali za predstavitev formalizacije učnih strategij v nadaljevanju. Štiri osnovne učne strategije in njihove lastnosti so prikazane v Tabeli 2.1 [5].

Tabela 2.1: Lastnosti štirih osnovnih učnih strategij.

Učna strategija	Vpletenost tutorja	Samostojnost učenca	Vodenje učenja	Vrsta komunikacije
Popolnoma usmerjen (angl. *tutor-driven*)	Velika	Ni samostojnosti	Tutor	Enosmerna od tutorja proti učencu
Nadzorovano (angl. *tutor-superveised*)	Srednja	Majhna	Tutor	Dvosmerna – učenje je za tutorja transparentno
Asistirano (angl. *tutor-assisted*)	Majhna	Srednja	Učenec	Dvosmerna – učenec prosi za pomoč
Popolnoma samostojno (angl. *tutor-unassisted*)	Ni vpletenosti	Velika	Učenec	Ni komunikacije

2.3 FORMALIZACIJA UČNIH STRATEGIJ

Da bi lahko učne strategije preučevali, jih moramo formalizirati. V tem podpoglavju si bomo pogledali, kako formalizirati strategije na primerih štirih strategij, kismo jih že predstavili.

Predpostavili bomo, da sta tutor in robot zmožna prej navedenih načinov komunikacije, da učni algoritem učenca zadostuje prej navedenim zahtevam. Ljudje lahko za opis svojega znanja o nečem uporabimo različne izraze, ki pomenijo različne stopnje znanja. Na primer, nekdo lahko svoje znanje programiranja oceni z dobro, če je nekoliko bolj samozavesten in izkušen, pa zelo dobro. Če je pri programiranju le začetnik, bo dejal, da je njegovo znanje slabo ali pa da ga sploh nima, če še ni nikoli programiral. Podobno oceno znanja naj bi bil sposoben opredeliti tudi učni algoritem. Glede na to, koliko je prepričan v svoje znanje, lahko poda »mehke labele« o nekem konceptu, ki ga trenutno proučuje. Torej poda labelo glede na stopnjo zaupanja v svoje znanje. Na primer, če poizkuša prepoznati nek naučeni koncept in je v pravilnost svoje prepoznave zelo prepričan, ji dodeli labelo, ki označuje ustrezno stopnjo zaupanja. Labele, pripadajoče stopnje zaupanjain njihove oznake prikazuje Tabela 2.2 [5].

Tabela 2.2: Mehke labele, njihove stopnje zaupanja in oznake.

Labele	Zaupanje v pravilno klasifikacijo koncepta	Oznaka
Da	Visoko zaupanje	YES
Verjetno da	Visoko zaupanje, a z dvomom	PY (Probably yes)
Verjetno ne	Nizko zaupanje	PN (Probably no)
Ne	Zelo nizko zaupanje	NO
Ne vem	Nezanesljivo prepoznavanje koncepta	DK (Don't know)
Neznano	Koncept, s katerim se prvič srečamo	UK (Unknown)

Ocena o lastnem znanju je potrebna zato, da se učenec lahko odloči, kaj bo storil s trenutnim znanjem, npr. ga posodobil ali vprašal tutorja za pomoč itd. Enako mora svojo sledečo akcijo na isti podlagi izbrati tutor, npr. ali naj pomaga učencu in v kakšni obliki itd. Učna strategija določi sledečo akcijo učenca na podlagi ocene lastnega znanja, ki jo učenec poda kot mehko labelo. Učna strategija določa tudi akcijo učitelja. Možne akcije učenca in tutorja, njihov opis in oznaka so prikazani v Tabeli 2.3 [5].

Tabela 2.3: Akcije učenca in tutorja, njihov opis in oznaka.

Akcija	**Opis**	**Oznaka**
Ne naredi ničesar	Učenec pusti svoje znanje kakršno je in ne sprašuje po pomoči tutorja.	/
Samodejno posodobi	Učenec spremeni svoje znanje glede na lastna opažanja, pri tem ne potrebuje pomoči tutorja.	U (Auto-update)
Povej	Tutor poda pravilno klasifikacijo samodejno, brez pobude učenca. Z njo učenec spremeni znanje.	T (Tell)
Vprašaj	Učenec prosi za pomoč tutorja, ta mu poda pravilno klasifikacijo. Z njo učenec spremeni znanje.	A (Ask)

Za akcije, ki zahtevajo spremembo znanja učenca, je treba definirati še, kakšna bo ta sprememba, ali se bo znanje nadgradilo ali pa popravilo. Sprememba znanja, njen opis in oznaka so prikazani v Tabeli 2.4 [5].

Tabela 2.4: Pozitivna in negativna sprememba znanja, njen opis in oznaka.

Sprememba znanja	**Opis**	**Oznaka**
Uporabi pozitivni primer	Učenec nadgradi svoje znanje s pomočjo pozitivnega primera koncepta.	**+**
Uporabi negativni primer	Učenec popravi svoje znanje s pomočjo negativnega primera koncepta.	_

Tutor in učenec lahko med seboj komunicirata na tri različne načine. V ta namen bomo za namen formalizacije učnih strategij definirali tri nivoje komunikacije med njima. Komunikacijski nivoji, njihov opis in oznaka so prikazani v Tabeli 2.5 [5].

Tabela 2.5: Nivoji komunikacije, njihovi opisi in oznaka.

Nivo komunikacije	**Opis**	**Oznaka**
Ignoriraj	Tutor se ne ozira na učenca. Tudi če učenec prosi za pomoč ali pride do napake pri učenju, tutor miruje.	ign (ignoring)
Poslušaj	Tutor posluša učenca. Ko ga ta prosi za pomoč, mu odgovori.	lst (listening)
Oceni	Tutor nadzira učenje učenca.	tfa (transparency facilitated assesment)

Za formalni zapis, ki uporablja mehke labele, akcije, načine posodabljanja znanja in različne komunikacijske nivoje, smo določili oznake. Te oznake bomo v nadaljevanju uporabili pri formaliziranem zapisu učnih strategij in ga tako naredili bolj preglednega.

Oznake akcij in pripadajoče posodobitve nastopajo skupaj. Na primer »*A+*« pomeni, da bo učenec vprašal tutorja, ta mu bo odgovoril s pozitivnim primerom, s katerim bo učenec posodobil znanje.

Tako kot od učnega algoritma zahtevamo, da ta zadostuje določenim zahtevam, tako zahtevamo tudi od tutorja, da vedno pozna pravilno interpretacijo koncepta (zlato pravilo), ki ga trenutno opazuje učni algoritem. Le tako lahko tutor pravilno pomaga učnemu algoritmu, če jo ta potrebuje.

Na podlagi do sedaj povedanega lahko definiramo učne strategije z vektorjem *LS* (angl. *learning strategy*), ki ga predstavljajo in definirajo enačbe (2.1), (2.2), (2.3), (2.4), (2.5) [5].

$$LS = \left[act_{sl,gt}\ , cl\right]\ , kjer \qquad (2.1)$$

$$sl \in \{YES, PY, PN, NO, DK, UK\} \qquad (2.2)$$

$$gt \in \{yes, no\} \qquad (2.3)$$

$$act_{\_,\_} \in \{\ /, U+, U-, T+, T-, A+, A-\ \} \qquad (2.4)$$

$$cl \in \{\ ign, lst, tfa\ \} \qquad (2.5)$$

LS je trinajstdimenzionalni vektor. Sestavljen je iz dvanajstdimenzionalnega vektorja *act* (2.4), ki nam opredeli obnašanje učenca in tutorja glede na različne stopnje zaupanja in pravilnost interpretacije. Za določitev akcije v vektorju *act* uporabimo množico *sl* (2.2),

kivsebuje možne stopnje zaupanja učenca v svoje znanje, in množico *gt (2.3)*, ta pa vsebuje možne vrednosti zlatega pravila.V praksi to pomeni, da učni algoritem ob opazovanju poskuša tega opredeliti. Upošteva tudi stopnjo zaupanja v svoje znanje in glede na to pripiše opazovanju mehko labelo. Natančno obnašanje tutorja in učenca pri interaktivnem učenju nam poda vektor $act_{sl,gt}$. Trinajsta dimenzija *LS* predstavlja komunikacijski nivo (2.5). Ta nivo se določi na podlagi vektorja *act* oziroma akcij, ki jih ta določa. Na primer, če učna strategija predvideva, da mora učitelj odgovarjati na učenčeva vprašanja, se komunikacijski nivo določi na *lst*. Poglejmo si primer zapisa vektorja v enačbi (2.6) [5].

$$LS = [T+, T-, A+, A-, T+,/, T+,/, T+,/, T+,/, lst] \qquad (2.6)$$

Vsaki dve zaporedni polji vektorja tvorita par, ki prikazujeta akciji ob določeni stopnji zaupanja v znanje učenca. Prvi par polj vektorja predstavljata najvišjo stopnjo zaupanja, četrti par pa najnižjo. Predzadnji par predstavlja akcije, če učenec svoje znanje oceni kot nezanesljivo, zadnji par pa, če učenec meni, da koncepta prej šeni srečal. Prvo polje v vsakem paru določa akcijo, če je interpretacija pravilna, drugo pa, če je ta napačna. Na koncu nam ostane zadnje, trinajsto polje, ki je rezervirano za določanje nivoja komunikacije.

2.3.1 Primeri formalizacije

Sedaj, ko smo predstavili potrebno formalizacijo, lahko v njej zapišemo prej spoznane štiri osnovne učne strategije.V prvem popolnoma nadzorovanem učenju tutor vodi celoten učni proces, ignorira robota, zato je komunikacijski nivo labeliran z *ign*. Tutor praktično uči učenca tako, da mu poda pravilno interpretacijo pozitivnega primera, s katerim učenec nadgradi znanje. Formaliziran zapis popolnoma nadzorovane učne strategije prikazuje enačba (2.7) [5].

$$LS_{td} = [T+,/, T+,/, T+,/, T+,/, T+,/, T+,/, ign] \qquad (2.7)$$

Pri nadzorovanem učenju je vektor učne strategije že nekoliko bolj zapleten. Ker mora tutor spremljati učni proces učenca, je nivo komunikacije med njima *tfa*. Če zazna napako učenca ob učenju, mu kot pri popolnoma nadzorovanem učenju poda pravilno interpretacijo primera, s katerim učenec nadgradi znanje (T+ ali T-). V nasprotnem primeru se v učenje ne umeša in dovoli učencu, da ta sam posodobi svoje znanje (U+). Formaliziran zapis nadzorovane učne strategije prikazuje enačba (2.8) [5].

$$LS_{ts} = [U+, T-, U+, T-, T+,/, T+,/, T+,/, T+,/, tfa] \qquad (2.8)$$

Asistirano učenje vsebuje že večjo vlogo učenca in manjšo vlogo tutorja, ki le čaka, da iniciativo k njegovem dejanju poda učenec. Zato je nivo komunikacije poslušaj (*lst*). Tukaj pride v poštev sposobnost učenca o sami oceni. Če učenec zaupa v pravilnost svoje interpretacije, bo znanje samodejno posodobil (U+), če pa v kakem primeru v svoje znanje ne

zaupa, pa prosi za pomoč poslušajočega tutorja, ki mu odgovori o pravilnosti njegove interpretacije. Če je učenec pravilno interpretiral, potem dobi pozitiven odgovor in svoje znanje nadgradi (A+), v nasprotnem primeru dobi negativen odgovor in svoje znanje popravi (A-). Formaliziran zapis asistirane učne strategije prikazuje enačba (2.9) [5].

$$LS_{ta} = [U+, U-, A+, A-, A+, A-,/,/, A+, A-, T+,/, lst] \quad (2.9)$$

V zadnjem, popolnoma samostojnem učenju že iz njegovega imena vidimo, da tutor ni potreben, zato je stopnja komunikacije nična (*ign*). Učenec tako v celoti vodi učni proces brez tutorja in v celoti sam posodablja svoje znanje glede na lastna opažanja. Formaliziran zapis popolnoma samostojne učne strategije prikazuje enačba (2.10) [5].

$$LS_{tu} = [U+, U+, U+, U+,/,/,/,/,/,/, T+,/, ign] \quad (2.10)$$

2.3.2 Formalizacija in iskanje novih učnih strategij

Formalizacija, predstavljena v tem poglavju, nam omogoča definiranje katerekoli poljubne učne strategije. Kot smo omenili v uvodu, je naš namen najti čimboljšo učno strategijo. Nove učne strategije bi lahko dobili že zelo preprosto tako, da bi naključno določili akcije v vektorju *act* in nato pogledali, kako se glede na te obnašata učitelj in učenec, ter izmerili, kako dobro poteka učenje. Lahko bi generirali vse možne kombinacije učnih strategij, ki nam jih omogoča zgornja formalizacija, in jih testirali. Tak pristop bi bil neučinkovit, saj bi zahteval testiranje 7^{12} možnih učnih strategij. V tem diplomskem delu je predlagan in uporabljen pristop iskanja učinkovite učne strategije s pomočjo genetskega algoritma. Ta algoritem in njegovo uporabo za naš namen bomo predstavili v naslednjihpoglavjih.

3 Genetski algoritem

Evolucijsko računanje izhaja iz navdiha po evolucijskem procesu razvoja živih bitij v naravi. Skozi generacije se živa bitja spreminjajo in prilagajajo svojemu okolju. Kako to dosegajo, je prvi razložil Charles Darwin [6], kjer je z naravnim izborom pojasnil ključni element tega procesa. Tisti posamezniki vrste, ki so bolje prilagojeni svojemu okolju, imajo večjo možnost, da v njem preživijo. Posledično imajo tudi večjo možnost imeti potomce. Tako se lastnosti, ki omogočajo preživetje, podajo naprej naslednji generaciji. Potomci imajo nekaj lastnosti od enega starša, nekaj pa od drugega, kar povzroči novo kombinacijo teh, ki bi bila lahko morebiti bolj uspešna. Lahko je tudi manj uspešna, v tem primeru ima posameznik manj možnosti za preživetje in manj možnosti, da svoje lastnosti poda naprej. Poleg tega obstaja še možnost mutacij. Te so naključne spremembe v lastnostih pri novih posameznikih. Če mutacije pomagajo k boljši prilagoditvi v okolju, se prek potomcev ohranijo v bazenu lastnosti. V nasprotnem primeru se škodljive mutacije izločijo, ker posamezniki z njimi težje preživijo. Selekcijo posameznikov, ki so najbolje prilagojeni, opravi okolje samo. S ponavljanjem tega procesa, selekcije, podajanja lastnosti potomcev in mutacije so vrste živih bitij vedno bolj prilagojene svojemu okolju.

Ta proces lahko posnemamo pri reševanju drugih problemov. Posameznik v generaciji predstavlja možne rešitve problema. Njihovo kakovost merimo z njihovo možnostjo prilagoditve okolju. Okolje pa je problem, ki ga rešujemo oziroma predstavlja naš cilj prilagoditve. Posameznik z najboljšo prilagoditvijo okolju bo najboljša rešitev našega problema [7].

Študijo prenosa prilagoditvenih procesov iz narave v umetne sisteme je prvi naredil John Holland [8]. Njegovo delo »Adaptation in Natural and Artificial Systems« vsebuje teoretično osnovo za prenos genetskega algoritma v računalniške sisteme. Holland je hotel najti nek splošen postopek, algoritem, ki bi bil primeren za reševanje širokega spektra problemov.

V tem poglavju bomo v prvem delu predstavili teoretično osnovo genetskih algoritmov, v drugem delu pa sledi razlaga, kako smo uporabili genetski algoritem za reševanje problema iskanja najboljše učne strategije.

3.1 Splošno o genetskem algoritmu

Splošno lahko predstavimo delovanje genetskega algoritma zelo preprosto. Začnemo z neko generacijo, s kombiniranjem najboljših posameznikov v njej naredimo novo generacijo in izvedemo proces mutacije. Nad novo dobljeno generacijo spet izvedemo izbor, naredimo potomce in izvedemo mutacije. To ponavljamo z vsako novo generacijo, dokler ne pridemo do želene rešitve. Ko se spustimo v podrobnosti, vidimo, da je treba odgovoriti na nekaj

ključnih vprašanj. In sicer, kako priti do začetne generacije, kako določiti najboljše posameznike in kako jih med seboj kombinirati, kako izvesti mutacijo ter kdaj zaključiti s celotnim procesom oziroma kdaj menimo, da smo prišli do rešitve in lahko algoritem zaključimo. Možne odgovore na ta vprašanja bomo predstavili v nadaljevanju. Najprej si bomo pogledali, kako predstaviti posameznika v kontekstu genetskega algoritma. Zelo pomembno je, da znamo izbrati najboljšo predstavitev, ki bo najbolj skladna našemu problemu.

3.1.1 Predstavitev posameznikov

Kandidatom za rešitev problema oziroma posameznikom znotraj generacije pravimo tudi genotipi. Kako jih bomo predstavili, je najbolj odvisno od narave problema, ki ga rešujemo. Najobičajnejše predstavitve posameznikov so binarna, permutacijska, reala in celoštevilska predstavitev. Katero izberemo, je odvisno od naše ocene, ki je najbolj primerna za naš problem, lahko pa jih med seboj tudi kombiniramo. Splošnega recepta za najbolj primerno predstavitev genotipa za neko vrsto problema ni, za izbiro se moramo zanašati na lastne izkušnje in znanje o problemski domeni ter genetskih algoritmih [7].

Binarna predstavitev

Ko je Holland prvič predstavil genetski algoritem, je uporabil binarno predstavitev genotipov. Tapredstavitev je zelo primerna za probleme, ki so že po svoji naravi binarni, na primer preklopne funkcije ali Boolova algebra. V binarni predstavitvi je mogoče predstaviti tudi ostale probleme tako, da nebinarne podatke prekodiramo v binarne [8].

Celoštevilčna predstavitev

Če so spremenljivke pri reševanju našega problema take, da lahko zavzamejo le celoštevilčne vrednosti, je verjetno celoštevilska predstavitev najprimernejša [7]. Za nas je celoštevilčna predstavitev pomembna, ker smo jo, kot bomo videli v nadaljevanju, uporabili za reševanje problema učnih strategij.

Realna predstavitev

Realna števila so znotraj računalnika predstavljena v plavajoči vejici. Njihova predstavitev je omejena z dolžino besede v računalniku, zato je tudi njihova natančnost omejena. Realna predstavitev se uporablja pri reševanju problemov, ki imajo rešitev v zvezni domeni [7].

Permutacijska predstavitev

Permutacijska predstavitevje primerna pri predstavitvi neke serije, vrstnega reda dogodkov. Uporabljata se dva različna načina za kodiranje permutacijske predstavitve [7]:

1. i-ti element permutacijske predstavitve predstavlja dogodek, ki se je zgodil na i-tem mestu začetne sekvence. Npr. začetna sekvenca (K,L,M,N) pri permutaciji (4,1,2,3) postane (N,K,L,M),
2. drugi način je vezan na prvega. Vzemimo spet začetno predstavitev (K,L,M,N), ki je pri prvem tipu permutacijske predstavitve (4,1,2,3) enaka sekvenci (N,K,L,M). Sedaj nam pri drugem načinu permutacija (4,1,2,3) določa še pozicijo i-tega elementa v sekvenci, ki smo jo dobili po permutaciji prvotne predstavitve prek prvega načina. Tako je končna predstavitev (M,N,K,L).

3.1.2 Inicializacija

Inicializacija genetskega algoritma pomeni generiranje začetne generacije. Ponavadi se uporabljata dva pristopa [7].

1. generirani genotipi v začetni generaciji so naključni oziroma vrednosti, ki jih zavzemajo, so naključne. Tako lahko zavzamejo vrednost v obsegu vseh možnih rešitev. S tem poizkušamo dobiti čimbolj raznolik bazen začetnih genotipov,
2. generirani genotipi so še vedno naključni, ampak poizkušamo njihov obseg vrednosti zožiti na množice možnih rešitev, kjer mislimo, da obstaja večja možnost končne rešitve.

Velikost začetne generacije je tako kot velikost sledečih generacij odvisna od problemske domene. Število posameznikov v njej je ponavadi od sto do nekaj tisoč genotipov. Večja začetna generacija pomeni, da smo za začetno iskanje rešitve zajeli večje območje možnih rešitev, v katerem bomo iskali končno rešitev. Premajhna generacija nam tako lahko že na začetku preveč zoža možnosti razvoja algoritma do rešitve. Preveč velika generacija pa lahko pomeni večjo možnost, da bi algoritem zašel v ne tako optimalna področja iskanja.

3.1.3 Selekcija

Selekcija je proces izbire prednikov oziroma tistih posameznikov, ki bodo stopili v proces reprodukcije. Osnova vsake selekcije je cenilna funkcija, s pomočjo katere ovrednotimo kakovost določenega posameznika. Bolj kakovostni posamezniki naj bi bili načeloma bolj primerni za reprodukcijo kot manj kakovostni. Pri selekciji želimo doseči, da se dobri genotipi prenesejo naprej, a paziti moramo, da s pretiravanjem v selekciji najkakovostnejših posameznikov lahko prezgodaj zožimo prostor iskanja rešitev algoritma. Temu pravimo prezgodnja konvergenca. Nekoliko slabši posameznik lahko pomeni neko novo pot do dobre rešitve, ki bo morda postala vidna komaj čez nekaj generacij, ko se bo genotip dovolj razvil. Zato je dobro, da kakovosti posameznika kot osnovo za selekcijo na nek način dodamo še faktor naključnosti, ki bo omogočil napredovanje v naslednjo generacijo tudi nekoliko manj kakovostnih, ampak še vedno perspektivnih genotipov. Poznamo več načinov selekcije, vsak

izmed njih poskuša na svoj način uravnotežiti zgoraj omenjene dejavnike, ki jih moramo upoštevati [7].

Selekcija s cenilko

Osnova selekcije je cenilna funkcija oziroma cenilka. Po oceni kakovosti vseh posameznikov lahko uporabimo preprosto enačbo, ki nam poda verjetnost za izbor posameznika. Verjetnost izbora i-tega posameznika označimo s p_i in vrednost njegove cenilke f_i. Velikost populacije je označena z μ. Enačba (3.1) prikazuje verjetnost izbire s cenilno funkcijo [7].

$$p_i = \frac{f_i}{\sum_{j=1}^{\mu} f_j} \qquad (3.1)$$

S tem načinom lahko zagotovimo, da bodo imeli najboljši osebki potomce. Pripelje pa lahko do drugih težav [7]:

1. če so genotipi približno enako kakovostni, bodo verjetnosti za njihov izbor približno enake, kar v bistvu privede do tega, da je izbor precej podoben naključnemu izbiranju,
2. lahko se zgodi, da pridemo v situacijo prehitre konvergence, ker najkakovostnejši posamezniki hitro izrinejo ostale manj kakovostnejše, a še vedno potencialno dobre genotipe.

Selekcija z rangiranjem

Vse genotipe najprej rangiramo glede na njihovo kakovost. Rang i-tega genotipa je označen z r_i. Genotipe razvrstimo od najmanj kakovostnegado najbolj kakovostnega, rang genotipa je nato njegov položaj v tem zaporedju. Enačba (3.2) nam podaja verjetnost izbire z rangiranjem [7].

$$p_i = \frac{r_i}{\sum_{j=1}^{n} r_j} \qquad (3.2)$$

Ruletna selekcija

Ruletna selekcija je dobila ime po načinu, po katerem si jo lahko predstavljamo. Posameznim genotipom dodelimo delež izseka na krogu sorazmerno z njihovimi verjetnostmi za izbiro. Verjetnost izbora posameznika izračunamo kot pri selekciji s cenilko ali rangiranjem. Nato generiramo naključno število, ki predstavlja točko v ruleti, kjer se ja ta ustavila. Izberemo posameznika, ki mu pripada površina izseka, kamor kaže točka [7].

Turnirska selekcija

Turnirska izbira je uporabna predvsem takrat, kadar cenilna funkcija zaradi različnih razlogov ne upošteva vseh faktorjev kakovosti pri svoji oceni, zato se opremo na relativno primerjanje dveh genotipov. Pare genotipov dobimo tako, da vsakega izberemo zaporedno vsaj enkrat,

temu pa vsakič izberemo drugega naključno izmed vseh ostalih genotipov. Nato primerjamo njihove kakovosti in izberemo tistega od njiju, ki je bolj kakovosten. To ponavljamo, dokler ne pridemo do konca izbire vseh genotipov [7].

3.1.4 Reprodukcija

Reprodukcija je proces v genetskem algoritmu, v katerem z operatorjema križanja in mutacije pridemo do novih genotipov.

Križanje

Po opravljeni selekciji kandidatov za reprodukcijo sledi korak križanja, kjer nastane nov posameznik. Naključno se izbereta dva starša in iz njiju napravimo potomce. Vrste križanja lahko razdelimo na tri skupine [7]:

1. enotočkovno križanje potrebuje točko, ki označuje točko križanja staršev. To točko generiramo naključno in lahko zavzema vrednosti od 0 do števila dolžine kode. Potomec tako dobi gene od prvega starša, ki se nahajajo pred mestom točke križanja, in gene drugega starša, ki se nahajajo za točko križanja,
2. n-točkovno križanje potrebuje več točk križanja. Z več točkami dobimo več segmentov genotipov staršev, ki jih alternirajoče združimo v novega potomca,
3. uniformno križanje obravnava vsak gen staršev posebej. Za vsak gen generiramo naključno vrednost, če je ta manjša kot 0,5, potem gen prvega preide v potomca, če pa je vrednost večja od 0,5, pa preide v potomca gen drugega starša.

Mutacija

Korak mutacije poskuša oponašati mutacijo v naravnem svetu. Z neko verjetnostjo spremenimo vrednost genotipa staršev ali potomcev. Ta verjetnost je običajno precej majhna, ponavadi toliko, da zagotovimo približno eno mutacijsko spremembo gena za vsako novo populacijo. Mutacija poskuša nekoliko povečati raznolikost genotipov, če so ti prepodobni, in tako morda lahko prepreči prehitro konvergiranje ali pa odpre možnost novih razvojnih poti. S tem namenom lahko mutacijsko funkcijo prilagodimo algoritmu tako, da se verjetnost mutacije poveča, če algoritem oceni, da se v zadnjih nekaj generacijah genotipi niso spremenili in rezultatsko stagnirajo [7].

3.1.5 Modeli populacij in izbira preživetja

Populacija igra vlogo že pri samem začetku genetskega algoritma, ko se odloča, kako velika bo začetna populacija in kako jo bomo generirali. Populacija je pomembna še pri izbiri prednikov in pri izbiri posameznikov za naslednjo generacijo. Pri generiranju naslednje generacije se postavita dve vprašanji. Kolikšen delež stare generacije se bo preneslo v novo generacijo in kateri genotipi bodo preživeli. Za tvorjenje nove generacije poznamo dva osnovna modela [7]:

1. stabilni model populacije deluje tako, da se z vsako novo generacijo zamenja le del stare generacije. Odstotek generacije, ki jo bomo zamenjali, kličemo tudi generacijska luknja. Če je μ število pripadnikov v generaciji in Ŋ število pripadnikov, ki jih zamenjamo, je generacijska luknja definirana kot Ŋ/μ,
2. generacijski model zahteva celotno zamenjavo generacije staršev z njihovimi potomci, ki smo jih dobili prek križanja in mutacije.

Zamenjava generacije nam pove, koliko prednikov in potomcev se prenese naprej in skupaj tvori novo generacijo. Pravilo elitizma teži k temu, da se vedno prenese v naslednjo generacijo najbolj kakovosten genotip v generaciji. V primerih velikih generacij, in če predpostavimo, da ne smemo podvajati kandidatov, se uporablja tudi obratni način elitizmu, to je način zamenjave najmanj kakovostnih genotipov [7].

3.1.6 Končanje genetskega algoritma

Na neki točki moramo genetski algoritem ustaviti. Ker problemi, ki jih rešujemo z genetskim algoritem, po naravi mogoče nimajo neke specifične rešitve, moramo določiti kriterije zaustavitve, pri katerih menimo, da smo prišli do dovolj dobre rešitve. Te kriterije lahko oblikujemo na več načinov. Najpreprostejši je določitev števila iteracij algoritma, po katerih se ta zaustavi. Lahko preverjamo, kdaj nek genotip doseže dovolj visoko kakovost, ki je za nas sprejemljiva in ne potrebujemo boljše rešitve, lahko pa poznamo najboljšo možno kakovost in je ta pogoj za končanje. Če algoritem že nekaj generacij bistveno ne izboljšuje genotipov, je lahko to tudi znak, da nadaljnje iskanje ne bo obrodilo sadov. Vedno pa obstaja tudi možnost, da algoritem ustavimo sami, ko se nam zdi to primerno. Zgoraj naštete pogoje za ustavitev lahko med seboj kombiniramo. Pri nekaterih problemih je določitev pogoja zaustavitve precej trivialna, pri drugih pa moramo za določitev pogojev uporabiti lastne izkušnje in znanje o problemski domeni ter genetskih algoritmih [7].

3.2 Optimizacija vektorjev učnih strategij z genetskim algoritmom

Osnovna ideja za optimizacijo vektorjev učnih strategij z genetskim algoritmom izhaja iz splošnosti genetskega algoritma za reševanje optimizacijskih problemov. Naš cilj je najti najboljšo učno strategijo, se pravi optimizirati vektor tako, da bo strategija, ki jo ta definira, najboljša možna.

3.2.1 Predstavitev genotipa

Učno strategijo smo v prejšnjem poglavju predstavili kot vektor oznak akcij in komunikacijskega nivoja. Od sedaj naprej bomo za predstavitev vektorjev uporabili drugačen format. Oznake bomo kodirali s celimi števili, ki so predstavljena v Tabelah 3.1 in 3.2.

Tabela 3.1: Kodiranje akcij učne strategije s celimi števili.

Oznake akcij	***Celoštevilske oznake***
/	*0*
U+	*1*
T+	*2*
A+	*3*
U-	*-1*
T-	*-2*
A-	*-3*

Tabela 3.2: Kodiranje komunikacijskega nivoja s celimi števili.

Oznake komunikacijskega nivoja	***Celoštevilske oznake***
ign	*0*
lst	*1*
tfa	*2*

Kot primere si lahko ogledamo, kako s celimi števili predstavimo vektorje štirih učnih strategij, kismo jih spoznali v prejšnjem poglavju. Enačbe ˙(3.3), (3.4), (3.5), (3.6) in (3.7) prikazujejo predstavitev znanih učnih strategij s celoštevilskim kodiranjem.

$$LS_{TD} = [T+,/,T+,/,T+,/,T+,/,T+,/,T+,/,ign]$$

$$LS_{TD} = [\,2\,,0\,,2\,,0,2,0,2,0,2,0,2,0,0] \qquad (3.3)$$

$$LS_{TS} = [U+,T-,U+,T-,T+,/,T+,/,T+,/,T+,/,tfa]$$

$$LS_{TS} = [\,1,-2,1,-2,2,0,2,0,2,0,2,0,2] \qquad (3.4)$$

$$LS_{TD} = [T+,/,T+,/,T+,/,T+,/,T+,/,T+,/,ign]$$

$$LS_{TD} = [\,2,0,2,0,2,0,2,0,2,0,2,0,0] \qquad (3.5)$$

$$LS_{TA} = [U+,U-,A+,A-,A+,A-,/,/,A+,A-,T+,/,lst]$$

$$LS_{TA} = [\,1,-1,3,-3\,,3,-3,0,0,3,-3,2,0,1] \qquad (3.6)$$

$$LS_{TU} = [U+,U+,U+,U+,/,/,/,/,/,/,T+,/,ign]$$

$$LS_{TU} = [\,1,1,1,1,0,0,0,0,0,0,1,0,0] \qquad (3.7)$$

Trinajsto polje vektorja predstavlja komunikacijski nivo. V nadaljevanju bomo videli, da smo naš problem poenostavili tako, da smo polje komunikacijskega nivoja postavili na vrednost, ki zahteva najvišje sodelovanje tutorja.Tako smo se izognili primeru strategije, kjer bi bilo smiselno, da tutor npr. posluša učenca, tega ne počne, ker je njegovo polje v trenutnem genotipu postavljeno na ignoriranje. Skratka, trinajsto polje komunikacijskega nivoja smo izločili iz procesa genetskega algoritma in tako preprečili generiranje določenih nesmiselnih genotipov.

3.2.2 Cenilna funkcija in selekcija

Do ocene kakovosti posameznika pridemo s cenilno funkcijo. V primeru vektorjev učne strategije to pomeni, da moramo učno strategijo, ki jo vektor definira, preizkusiti. Cenilna funkcija požene interaktivno učenje med učencem in tutorjem, testira naučeno in vrne rezultate tega testiranja. Ti rezultati naučenega so naša ocena kakovostnega genotipa. Ko imamo oceno kakovosti za vse genotipe v generaciji, jih razvrstimo od najmanj do najbolj kakovostnega. Za merjenje kakovosti genotipa smo uporabili dve meri, uspešnost prepoznavanja in ceno tutorstva [5].

Uspešnost prepoznavanja

Uspešnost prepoznavanja izračunamo glede na uspešnost prepoznanih predhodno naučenih konceptov pri interaktivnem učenju. Za izračun uspešnosti prepoznavanja se upremo na zlato pravilo oziroma »ground truth« podatke. Pri pravilni presoji, ali primer spada v naučeni koncept, ne točkujemo prepoznavanja pozitivno, pri napačni presoji pa točkujemo negativno. V Tabeli 3.3 predstavljamo mehko točkovanje, ki je vezano na zaupanje klasifikacije konceptov in njihovo pravilnost [5].

Tabela 3.3: Točkovanje mehkih label glede na pravilnost koncepta.

	DA	**Verjetno da**	**Verjetno ne**	**Ne**	**Ne vem**	**Neznano**
Da	1	0,5	–0,5	–1	0	0
Ne	-1	–0,5	0,5	1	0	0

V prvi vrstici so predstavljene možne vrednosti zaupanja v prepoznavanje koncepta v levem stolpcu pa »ground truth« podatki oz. zlato pravilo. Iz tabele vidimo, da ob pravilni prepoznavi in visoki stopnji zaupanja v prepoznavo dodelimo več točk, obratno jih ob napačni prepoznavi odbijemo.

Cena tutorstva

Cena tutorstva je različna glede na to, kakšno vrsto interakcije imamo med učencem in tutorjem. Če mora tutor sam neprestano bedeti nad učenjem učenca in ga popravljati ob napaki, je ta cena večja, kot pa takrat, kadar mora učenec sam prositi za pomoč. Najmanjša cena tutorja je takrat, kadar ta sploh ni potreben pri učnem procesu, največja pa je takrat, kadar mora sam voditi celotno učenje. Cena tutorstva je tako obtežena vsota različnih akcij, ki jih izvede tutor med interaktivnim učenjem. Tutor lahko naredi dve akciji: prva je ta, da daje učencu pravilne informacije o nekem konceptu. Če pogledamo štiri osnovne strategije, se to dogaja v primeru popolnoma usmerjenega učenja, kjer tutor uči učenca. Ta akcija se uporabi tudi pri nadzorovanem učenju, kjer tutor poda pravilno informacijo, če zazna, da se je pri samostojnem učenju učenec nekaj narobe naučil. Druga akcija je odgovor na vprašanje učenca. To se zgodi v načinu »tutor-assisted«, kjer tutor le pomaga z odgovorom na učenčevo vprašanje. Glede na kompleksnost akcije ju lahko primerno obtežimo. Pri podajanju informacije pa je potrebno s strani tutorja več truda, saj ima tutor večjo vlogo in vodi učenje oz. nadzira učenca, zato je to akcijo primerno več obtežiti kot pri akciji odgovarjanja na vprašanje. Pri odgovarjanju tutorju ni treba voditi učenja, temveč le odgovarjati, ko je to potrebno, zato je primerna obtežitev manjša. Tabela 3.3 prikazuje točkovanje akcij tutorja [5].

Tabela 3.3: Tutorjeve akcije med interaktivnim učenjem in njihove obtežitve

Tutorjeva akcija	**Utež**
Podaj informacijo	1
Odgovori na vprašanje	0,25

Tako lahko zapišemo enačbo (3.8) za izračun cene tutorstva (angl. *tutpring cost*) kot obteženo vsoto števila podanih informacij in števila podanih odgovorov [5].

$$TC = N_{inf}\, C_{inf} + N_{ans}\, C_{ans} + C_{cl} \qquad (3.8)$$

Tukaj je treba opozoriti, da akcije, ki ne zahtevajo vpletenosti tutorja, nima smisla upoštevati v zgornji enačbi, saj bi bila njihova obtežitev enaka nič. Taka akcija bi bila akcija »ignoriraj«. Zgornja enačba vsebuje tudi člen C_{cl}, ki prišteva vrednost komunikacijskega nivoja. Smiselno je, da višji komunikacijski nivo med učiteljem in tutorjem pomeni večjo ceno tutorja. V drugem poglavju smo omenili, da smo zaradi enostavnejše implementacije komunikacijski nivo strategij, ki smo jih preiskovali, vedno postavili na najvišjo vrednost. To pomeni, da bi v naših preiskanih strategijah za enačbo 3.8 vedno pripisali isto konstantno vrednost C_{cl}-ja. Praktično bi cenam tutorstva naših strategij vedno prišteli enako vrednost, zato je bilo smiselno, da smo ta člen iz enačbe izpustili.

Kombinirana mera

Kot smo že omenili, je treba za končno oceno upoštevati tako uspešnost prepoznavanja in ceno tutorstva. Za ta namen smo uporabili razmerje med njima in jo poimenovali kombinirana mera, katere enačbo (3.9) vidimo spodaj.

$$Kombinirana\ mera = \frac{Uspešnost\ prepoznavanja}{Cena\ tutorstva} \qquad (3.9)$$

Kombinirana mera bo velika takrat, kadar bo število pravilno prepoznanih primerov koncepta pravilno, tako uspešnost pa bomo dosegli pri interaktivnem učenju ob čim manjšem posredovanju tutorja.

3.2.3 Križanje in mutacija

V genetskem algoritmu je uporabljeno enotočkovno križanje. Točko križanja lahko sami določimo ali pa izberemo naključno izbiro točke za vsako križanje posebej. Izbor dveh staršev lahko poteka na dva načina, in sicer sta lahko starša določena naključno ali pa po načinu, da vsak genotip križamo z vsemi drugimi vsaj enkrat, odvisno od velikosti generacije. Na Sliki 3.1 je primer križanja dveh vektorjev in nastanek njunega potomca, če je točka križanja peto polje vektorjev.

$$LS_{TD} = [2,0,2,0,2,0,2,0,2,0,2,0,2]$$

$$\times$$

$$LS_{TU} = [1,1,1,1,0,0,0,0,0,0,1,0,0]$$

$$\downarrow$$

$$LS_{new} = [2,0,2,0,2,0,0,0,0,0,1,0,0]$$

Slika 3.1: Primer križanja dveh vektorjev.

Ko imamo enkrat potomce, nad njimi izvedemo še korak mutacije. Za vsako polje v vektorju generiramo naključno število, glede na katero izvedemo mutacijo ali ne. Vrednosti vektorja lahko mutirajo na dva načina. Dovolimo jim le majhne spremembe v vrednosti, torej neka vrednost lahko mutira le v njej prvo najbližjo večjo ali manjšo vrednost. Drugi način pa omogoča, da lahko neka vrednost mutira v katerokoli drugo možno vrednost. Na Sliki 3.2 je primer mutacije na prvem polju vektorja.

$$LS_{new} = [2,0,2,0,2,0,0,0,0,0,1,0,0] \rightarrow LS_{new}{}' = [3,0,2,0,2,0,0,0,0,0,1,0,0]$$

Slika 3.2: Mutacija na prvem polju vektorja.

3.3.3 Postopek optimizacije in zaključek genetskega algoritma

Po opisu genetskih algoritmov in formalizaciji učnih strategij lahko nakažemo, kako bi lahko z genetskimi algoritmi optimizirali učne strategije.

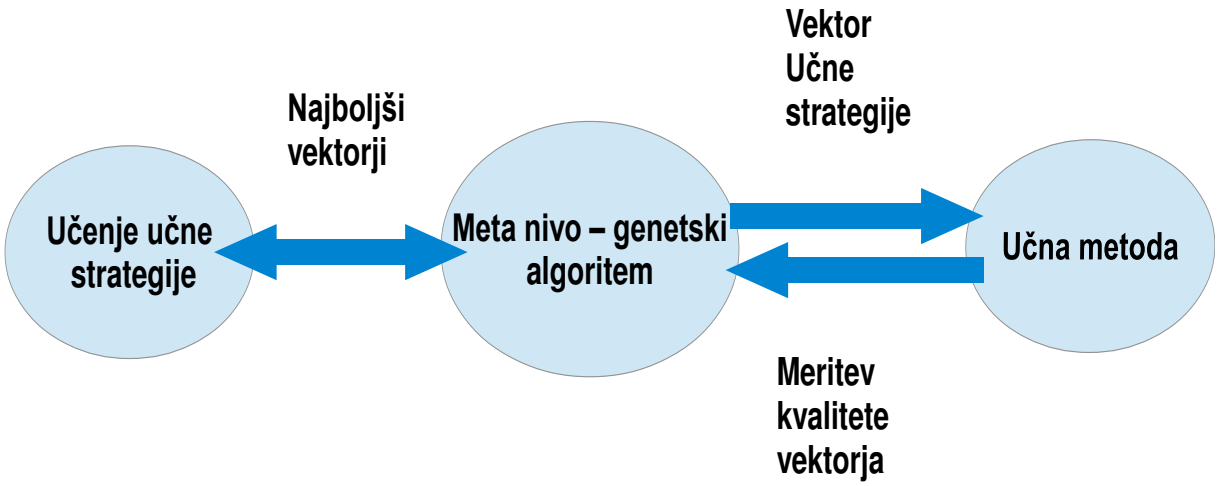
Naše posameznike v generaciji bi predstavljali vektorji učnih strategij s prekodiranimi akcijami v cela števila. Te posameznike bi ocenili s cenilno funkcijo. Funkcija bi pokazala, kako učinkovita je neka učna strategija. Najbolj kakovostne vektorjebi nato med seboj križali in nad potomci izvedli mutacijo,tako pa bi dobili novo generacijo. Postopek bi ponavljali, dokler ne bizadostili nekemu kriteriju zaustavitve. Konkretno smo za kriterij uporabili določeno število generacij, po katerih se je algoritem zaključil. Natančnejšo implementacijo in izvedbo algoritma si bomo pogledali v sledečem poglavju.

4 Programska implementacija

Program preiskovanja učnih strategij z genetskim algoritmom za izvajanje učnega procesa uporablja program za simuliranje interaktivnega neprestanega učenja oziroma ICLS (angl. *Interactive Continuous Lerning Simulator*) [5]. V kodi ICLS-ja se tako izvaja učni algoritem interaktivnega učenja, ki se zažene vsakič, ko testiramo določeno učno strategijo. To se zgodi, ko se zažene cenilna funkcija in hoče oceniti kakovost neke učne strategije oziroma vektorja, ki jo predstavlja.

4.1 Učna metoda

Učenje učne strategije poteka prek genetskega algoritma. Naučene učne strategije so najboljši vektorji generacije. Ko se genetski algoritem zaključi, so končne naučene učne strategije najboljši vektorji te zadnje generacije. Genetski algoritem tako deluje kot vmesnik med učnimi metodami, ki služijo učenju učenca, in naučenimi najboljšimi učnimi strategijami, kar prikazuje Slika 4.1.



Slika 4.1: Vloga učne metode pri učenju učnih strategij

Učni algoritem mora biti sposoben najti povezave med vizualnimi značilnostmi in vrednostim atributov. Pri vseh slikah, ki imajo isto vizualno značilnost, mora določiti specifični vizualni atribut za to značilnost. Učni algoritem izbere vizualno značilnost vseh slik glede na to, katera značilnost je najbolj prisotna v vseh slikah. Istočasno preveri, da ni neka druga značilnost prisotna pri vseh slikah v enaki meri. Povprečna vrednost in varianca najboljše značilnosti predstavljata vrednost vizualnega atributa. Tako zagotovi edinstvenost tega atributa [9].

4.1.1 Začetno učenje

Učenje se začne tako, da programu predložimo sklop slik, s katerih bo pridobil začetno znanje, ki ga bo kasneje lahko tudi posodabljal. Za vsako sliko moramo predložiti tudi seznam njihovih atributov, kot so npr. barva, oblika, velikost itd. [9].

Algoritem:

Vhod: Sklop učnih slik $\boldsymbol{X}$, seznam vrednosti atributov $\boldsymbol{AV_i}$ za pripadajočo učno sliko$\boldsymbol{X_i}$.
Izhod: Modeli vrednosti atributov $\boldsymbol{mAV_i}$, $\boldsymbol{i = 1 \ldots N_{av}}$.

1. Za vsak $\boldsymbol{AV_i}$,$\boldsymbol{i = 1 \ldots N_{av}}$naredi:
2. izvleci vse vizualne značilnosti $\boldsymbol{F_j}$, $\boldsymbol{j = 1 \ldots N_f}$ iz vsake učne slike v $\boldsymbol{X}$,
3. najdi skupino slik $\boldsymbol{X_i}$, ki vsebujejo vse slike, labeliarne z $\boldsymbol{AV_i}$,
4. izračunaj povprečja in variance za vrednosti od vsake značilnosti $\boldsymbol{F_j}$ čez vse slike v $\boldsymbol{X_i}$,
5. zaključi zanko,
6. izračunaj ***min*** in ***max*** vseh vrednosti vsake značilnosti $\boldsymbol{F_j}$,
7. normaliziraj vse variance s pridobljenimi intervali vrednosti značilnosti: $\boldsymbol{nvar_{ij} = var_{ij} \,/\, (maxVar_j - minVar_j)^2,\ i = 1 \ldots N_{av},\ j = 1 \ldots N_f.}$
8. Za vsak $\boldsymbol{AV_i}$,$\boldsymbol{i = 1 \ldots N_{av}}$ naredi:
9. izberi značilnost $\boldsymbol{F_j}$ z najmanjšo normalizirano varianco $\boldsymbol{nvar_{ij}}$,
10. shrani povprečje in varianco izbranega $\boldsymbol{F_j}$, tako da oblikuješ $\boldsymbol{mAV_i}$, kar predstavlja model za $\boldsymbol{AV_i}$,
11. zaključi zanko.

4.1.2 Nadgrajevanje znanja

Ko imamo osnovno znanje, naučeno iz predloženih slik, ga lahko nadgrajujemo. Znanje nadgrajujemo tako, da eno za drugo opazujemo vsako naslednjo novo sliko [9].

Algoritem:

Vhod: Modeli vrednosti atributov $\boldsymbol{mAV_i}$, $\boldsymbol{i = 1 \ldots N_{av}}$, statistika značilnosti $\boldsymbol{FS}$, nova slika $\boldsymbol{X}$ s pripadajočo vrednostjo atributov $\boldsymbol{AV_i}$. **Izhod:** Posodobljen $\boldsymbol{mAV_i}$, $\boldsymbol{i = 1 \ldots N_{av}}$ in $\boldsymbol{FS}$.

1. Če model za $\boldsymbol{AV}$ še ni bil naučen, ga inicializiraj,
2. posodobi povprečje in varianco značilnosti v odnosi za $\boldsymbol{AV}$ (shranjen v $\boldsymbol{FS}$),
3. posodobi vse ***min*** in ***max*** za značilnosti,
4. nadaljuj s koraki od 7 do 11 algoritma za začetno učenje.

4.1.3 Prepoznavanje

Spodnji algoritem je sposoben prepoznavati vizualne značilnosti novega objekta na sliki s pomočjo prej naučenih modelov vizialnih značilnosti. Glede na to, koliko je opazovani objekt blizu prej naučenim modelom, lahko algoritem odgovori z različnimi odgovori [9].

Algoritem:

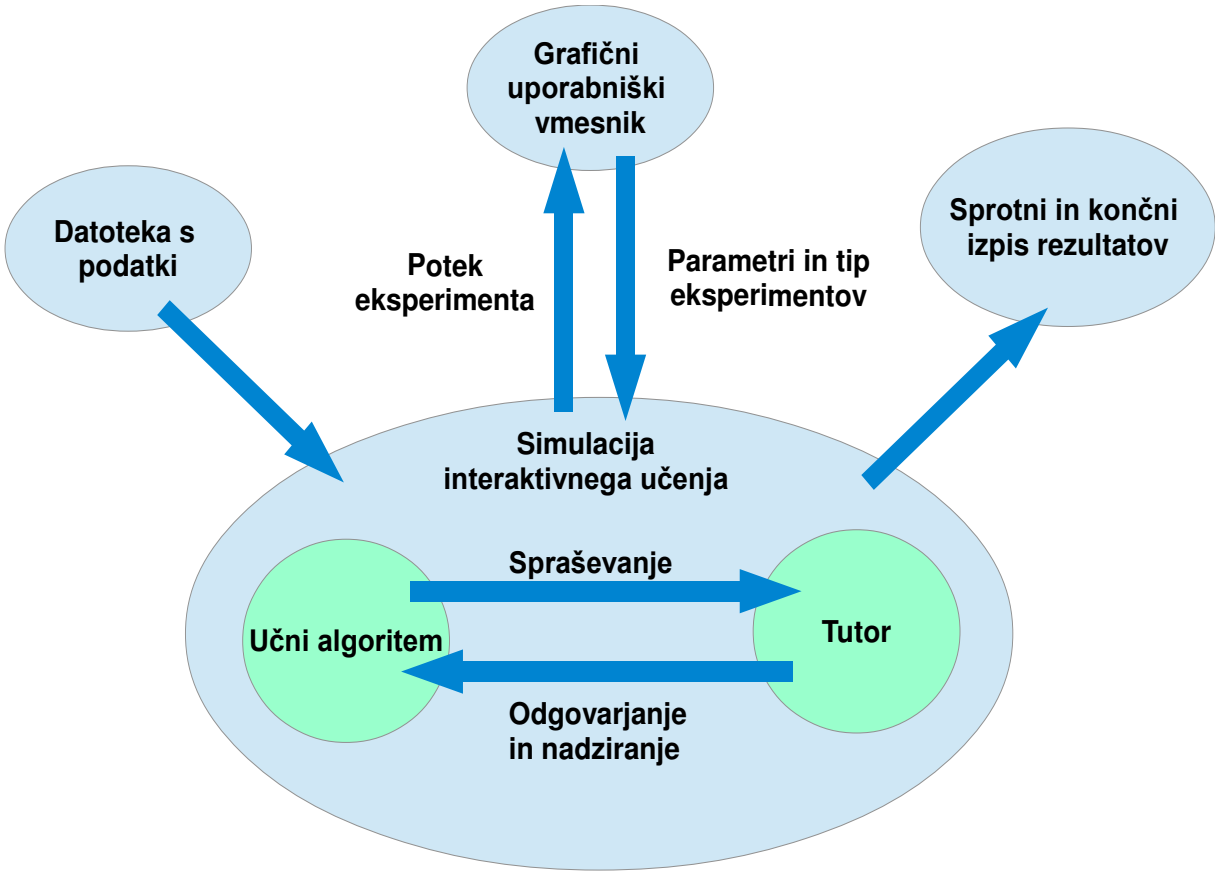
Vhod: Slika $\boldsymbol{X}$, vprašanje »Ali je to $\boldsymbol{AV}$?«, kjer je $\boldsymbol{AV}$ nek atribut.
Izhod: Odgovor na vprašanje »Ali je to $\boldsymbol{AV}$«.

1. Če mode za $\boldsymbol{AV}$ še ni bil naučen, odgovori z »Ne vem«,
2. določi, katera značilnost $\boldsymbol{F_j}$ je povezana z vrednostjo atributa $\boldsymbol{AV}$ v modelu $\boldsymbol{mAV}$,
3. izvleci vrednost te značilnosti $\boldsymbol{F_j}$iz slike $\boldsymbol{X}$,
4. izračunaj $\boldsymbol{d = (F_j - mAv.mean) / \sqrt{mAv.var}}$.,
5. če $\boldsymbol{d \in [0 , T_{da}]}$, odgovori z »**Da**«,
6. če $\boldsymbol{d \in [T_{da}, T_{vd}]}$, odgovori z »**Verjetno, da**«,
7. če $\boldsymbol{d \in [T_{vd}, T_{vn}]}$, odgovori z »**Verjetno, ne**«,
8. če $\boldsymbol{d \in [T_{vn}, , T_{ne}]}$, odgovori z »**Ne**«.

Metoda, ki jo uporabljamo za učenje, nima implementiranega popravljanja napačno naučenega znanja oziroma »unlearninga«. Uporaba algoritma z implementiranim »unlearningom« bi bila preveč zahtevna za evaluacijo.

4.2 Simulator interaktivnega neprestanega učenja

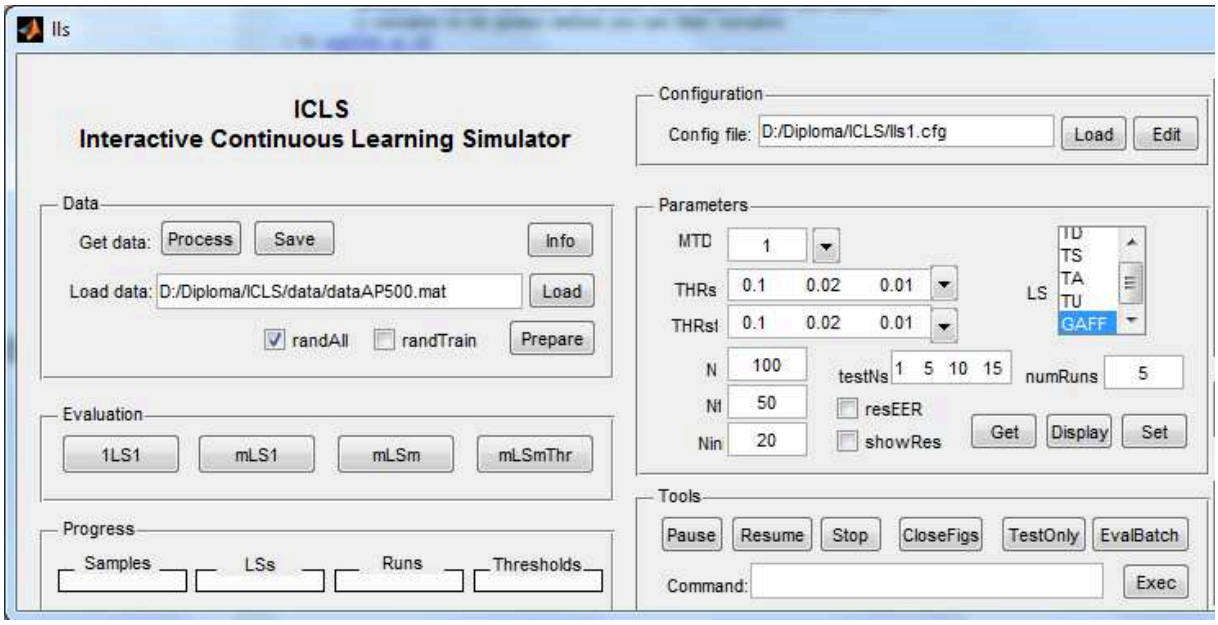
Zavedati se moramo, da je osnova za vsa testiranja v programu za preiskovanje učnih strategij interaktivni učni algoritem, ki je del Simulatorja interaktivnega neprestanega učenja (angl. *Interactive Continuous Learning Simulator – ICLS*). Poleg učnega algoritma je pomemben tudi del programa, ki simulira vsevednega tutorja, ki sodeluje pri učnem procesu z učnim algoritmom. ICLS je bil razvit za potrebe testiranja in vrednotenja učnih strategij [5]. Programske dele ICLS-ja prikazuje Slika 4.2.



Slika 4.2: Programski deli ICLS-ja.

4.2.1 Grafični uporabniški vmesnik ICLS-ja

V grafičnem uporabniškem vmesniku ICLS-ja lahko nastavimo parametre učnega algoritma. Ta algoritem lahko izvajamo tudi samostojno, se pravi brez preiskovanja strategij z genetskim algoritmom. Slika 4.3 prikazuje grafični uporabniški vmesnik ICLS-ja [5].



Slika 4.3: Grafični uporabniški vmesnik ICLS-ja

Grafični vmesnik lahko razdelimo na več segmentov. V segmentu »Parameters« nastavljamo parametre učnega algoritma, ki so:

- DMT – padajoči meni, s katerim izberemo eno izmed treh učnih metod, ki so Gaussian of Best Feature (GBF), Kernel Density of Best Feature (KDBF) in Multidimensional Kernel Density of Best Feature (MKDBF),
- THRs in THRSt – padajoča menija, s katerima določimo pragove za učenje in testiranje,
- N – tekstovno polje, v katerem določimo število učnih slik,
- Nt – tekstovno polje, v katerem določimo število testnih slik,
- Nin – tekstovno polje, v katerem določimo število slik, ki se bodo obdelale z začetno učno strategijo,
- testNs – tekstovno polje, v katerem določimo, po katerih učnih slikah naj se izvede test,
- numRuns – tekstovno polje, v katerem določimo število ponovitev istega eksperimenta,
- resERR – izbirno polje, ki določa, ali se na koncu izvede računanje optimalnega praga (pri »equal error rate«),
- showRes – izbirno polje, ki določa, ali naj se izrisujejo rezultati,
- LS – izbirno okno, v katerem določimo učno strategijo, ki jo želimo testirati. Vsebuje pet možnosti: prve štiri TD, TS, TA in TU določajo osnovne štiri učne strategije. Izbira GAFF (Genetic Algorithm Fit Function) določi, da bo učni algoritem uporabljen v genetskem algoritmu. Če to izbiro poženemo samostojno znotraj ICLS-ja, bo testirana strategija določena v polju za predefinirane vektorje genetskega algoritma,

- Set – gumb, s katerim potrdimo nastavljene parametre,
- Display – gumb, s katerim prikažemo trenutno nastavljene parametre,
- Get – gumb, s katerim preberemo trenutno nastavljene parametre v globalnih spremenljivkah.

V segmentu »Tools« so različna orodja:

- Pause – gumb, s katerim začasno ustavimo izvajanje simulacije,
- Resume – gumb, s katerim ponovno zaženemo simulacijo,
- Stop – gumb, s katerim ustavimo simulacijo,
- CloseFigs – gumb, s katerim zapremo vse slike rezultatov,
- TestOnly – gumb, s katerim poženemo samo evaluacijo brez učenja,
- EvalBach – gumb, s katerim poženemo paketno testiranje na vseh podatkih hkrati,
- Exec – gumb, s katerim v mirovanju simulacije poženemo ukaz, vnešen v tekstovno polje »Command«.

Segment »Data« služi za nalaganje podatkov:

- Load – gumb, s katerim naložimo podatke, za katere nastavimo pot v tekstovnem polju Load Data,
- Prepare – gumb, s katerim pripravimo na novo naložene podatke,
- Info – gumb, s katerim prikažemo podatke o podatkih,
- Process in Save – gumba sta namenjena dejanskemu procesiranju slik in pridobivanju podatkov iz njih.

V segmentu »Configuration« določimo konfiguracijsko datoteko, v kateri definiramo vse privzete parametre:

- Load – gumb, s katerim naložimo konfiguracijsko datoteko, na katero kaže pot v tekstovnem polju Load File,
- Edit – gumb, s katerim lahko urejamo trenutno konfiguracijsko datoteko.

V segmentu »Evaluation« poženemo različne tipe eksperimentov:

- 1LS1 – gumb, s katerim enkrat evaluiramo izbrano učno strategijo,
- m1LS1 – gumb, s katerim enkrat evaluiramo več izbranih učnih strategij (v izbirnem oknu LS),

- mLSm – gumb, s katerim evaluiramo več izbranih učnih strategij, kolikor krat je določeno v tekstovnem polju numRuns,
- mLSmThr – gumb, s katerim evaluiramo kot pri mSLm, s to razliko, da upošteva še v naprej določene pragove, vendar pri različnih naborih parametrov.

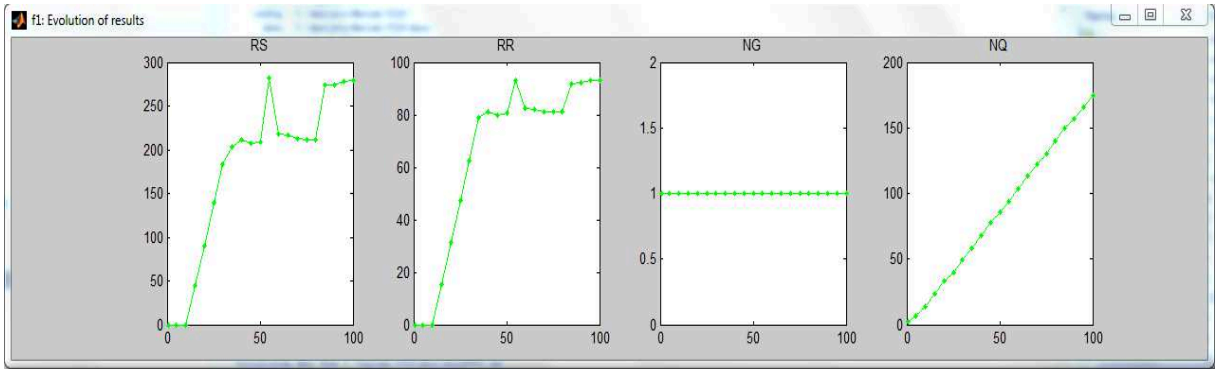
V segmentu »Progress« lahko spremljamo napredek pri izvajanju operacij iz segmenta »Evaluation«.

4.2.2 Prikaz rezultatov ICLS-ja

Pri izbiri eksperimenta 1LS1 se v oknu skozi čas izrisujejo sledeči rezultati:

- RS – Recognition Score (uspešnost prepoznavanja),
- RR – Recognition Rate (uspešnost prepoznavanja),
- NG – Number of Gaussians oziroma components (kompleksnost modela),
- NQ – Number of Questions oziroma tutoring cost (odvisno od implementacije).

Na Sliki 4.4 vidimo izris rezultatov, ki prikazujejo spremembo različnih mer skozi čas [5].

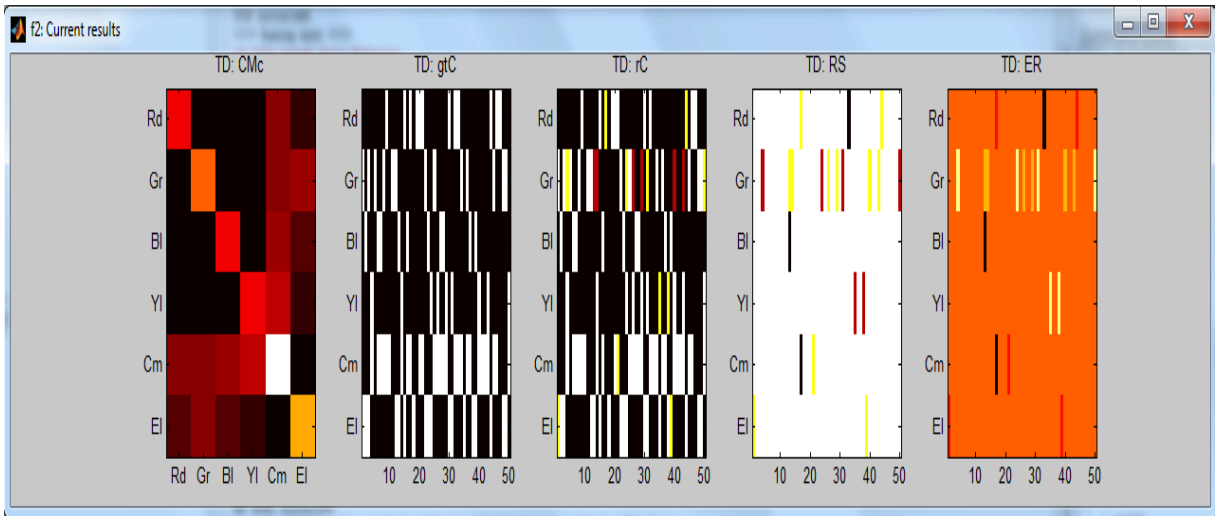


Slika 4.4: Izris spremembe mer skozi čas.

Na koncu se izrišejo še končni rezultati:

- CMc – Confusion Matrix,
- gTC – Ground Truth (zlato pravilo): koncepti za posamezno sliko,
- rC – razpoznani koncepti,
- RS – Recognition score za posamezno sliko,
- ER – napaka.

Na Sliki 4.5 vidimo izris končnih rezultatov [5].



Slika 4.5: Končni rezultati ICLS-ja.

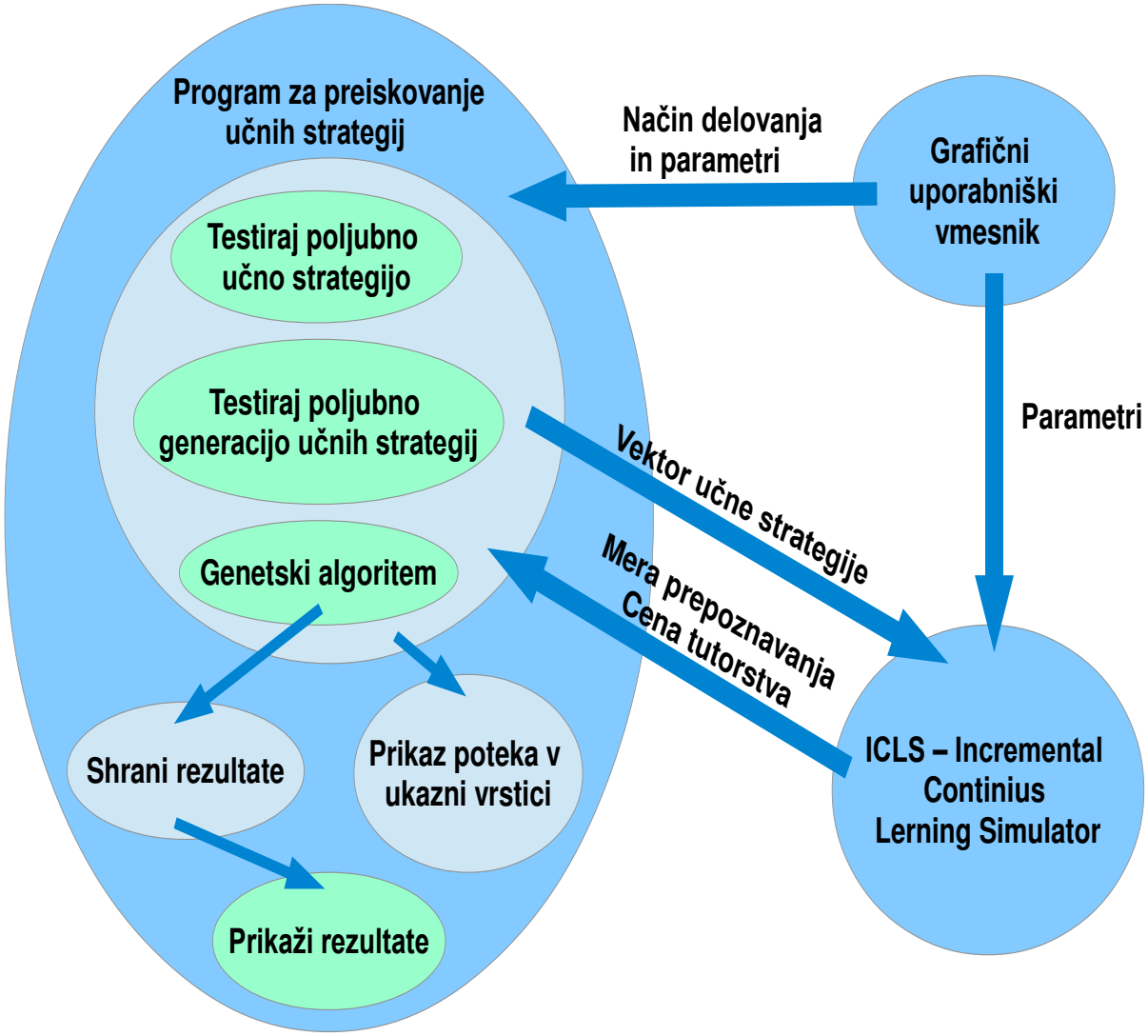
- Prva matrika CMc je matrika zamenjav. Pove, koliko primerov posameznega koncepta, podanega v vrsticah, se preslika v katere koncepte, podane v stolpcih.
- Druga matrika z leve je matrika gtC oz. matrika zlatega pravila (angl. *ground truth*). Testni primeri so podani v stolpcih. Če ta primer pripada dotičnem konceptu, je v posamezni vrstici vrednost 1 (bela) ali 0 (črna), če mu ne pripada.
- Sredinska matrika rC. Testni primeri so podani v stolpcih. Na podoben način kot tabela gtC predstavi, kateri koncepti so bili razpoznani pri posameznih testnih primerih.
- Druga matrika z desne predstavlja mero prepoznavanja (angl. *recognition score*) za posamezen testni primer.
- Zadnja desna matrika pa predstavlja napako za posamezni testni primer oziroma razliko med matriko rC in gtC. Če izberemo možnosti z več eksperimenti, se izrisujejo povprečni rezultati.

4.3 Program za preiskovanje učnih strategij

Celotni program preiskovanja učnih strategij vsebuje tudi možnost klica le nekaterih posameznih funkcionalnosti, ki jih sicer uporabljamo v genetskem algoritmu. Ločimo lahko štiri načine delovanja programa:

- genetski algoritem za preiskovanje učnih strategij,
- testiranje poljubne učne strategije,
- testiranje poljubne generacije učnih strategij,
- grafični prikaz rezultatov genetskega algoritma učnih strategij.

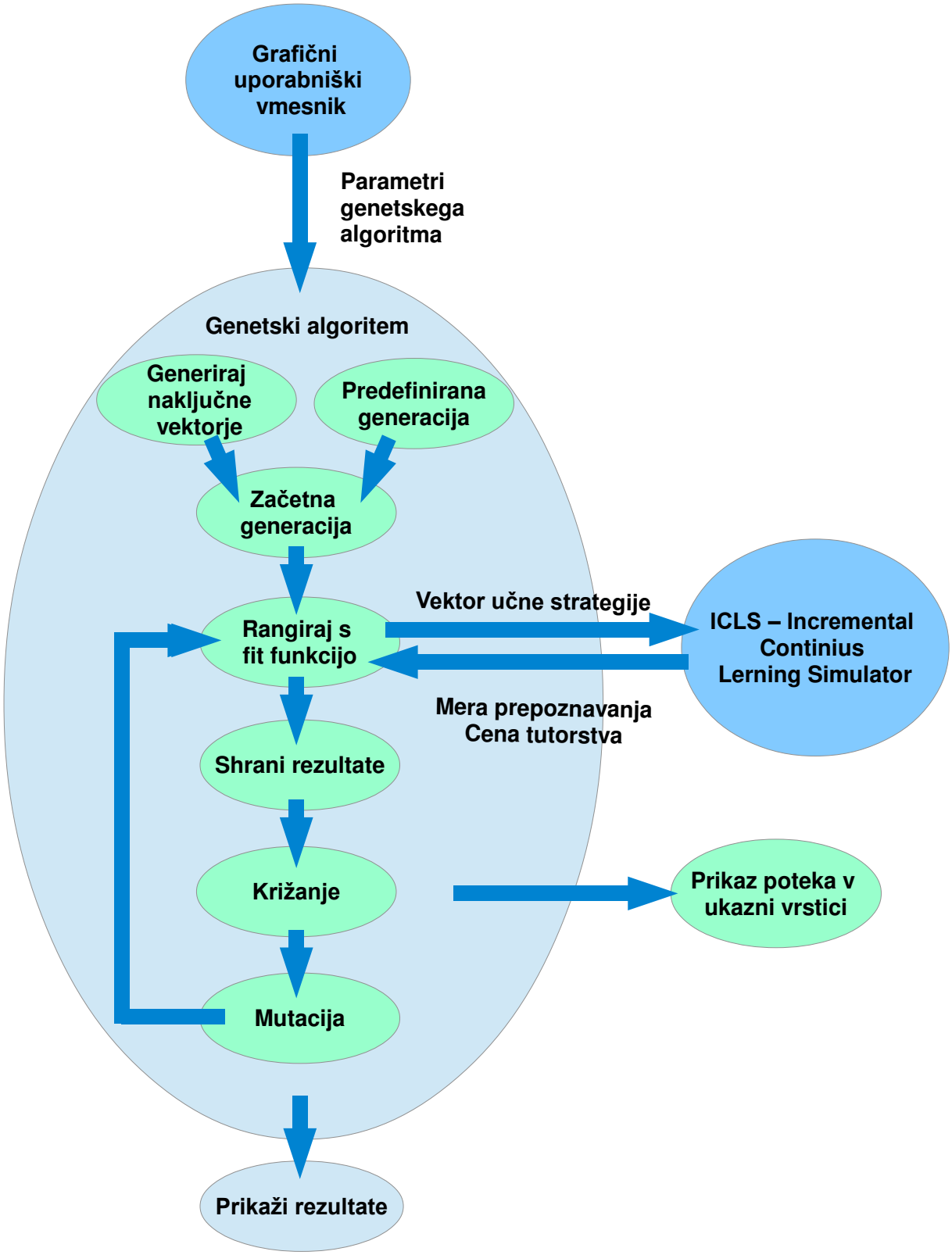
Program upravljamo prek grafičnega uporabniškega vmesnika. Prek njega zaženemo izbrani način delovanja in nastavimo potrebne parametre. Med delovanjem lahko spremljamo potek prek ukazne vrstice, kjer se izpisuje stanje programa. V načinu za testiranje poljubne učne strategije ali generacije učnih strategij se v ukazni vrstici na koncu izpišejo tudi rezultati, v primeru genetskega algoritma pa se rezultati shranijo v določeno datoteko. Grafični prikaz nato uporabi datoteko z rezultati in jih prikaže v obliki grafov. Na Sliki 4.6 vidimo interakcije med posameznimi deli programa.



Slika 4.6: Interakcije med posameznimi deli programa

4.4 Implementacija genetskega algoritma

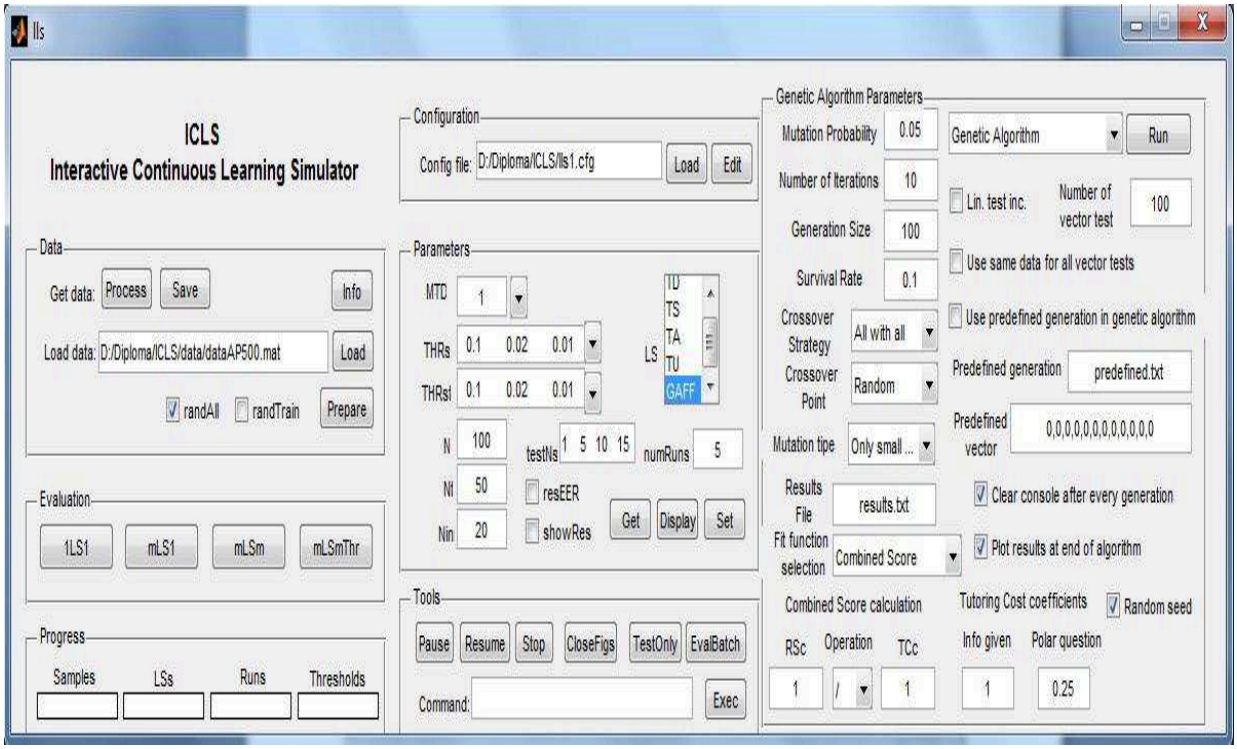
Genetski algoritem se zažene s pomočjo grafičnega uporabniškega vmesnika. Z njegovo pomočjo se pošljejo tudi potrebni parametri za delovanje genetskega algoritma. V prvem koraku algoritma se generira naključna generacija ali pa vzame predefinirano generacijo iz določene datoteke. Tukaj algoritem vstopi v iteracijsko zanko generacij. V prvem koraku se s cenilno funkcijo, ki uporabi funkcionalnost ICLS-ja, preveri kakovost vektorjev v posamezni generaciji in jih primerno rangira. Nato vektorji vstopijo v korak križanja in na zadnje v korak mutacije. Po vsakem rangiranju se najboljši vektorji vedno shranijo v datoteko z rezultati. Med delovanjem se nam v ukazni vrstici izpisuje potek posamezne iteracije in nekoliko podrobneje potek testiranja cenilne funkcije. Ko se algoritem zaključi, lahko prikaže prej shranjene rezultate v obliki grafov, ki jih izriše na zaslon. Na Sliki 4.7 vidimo, kako poteka genetski algoritem.



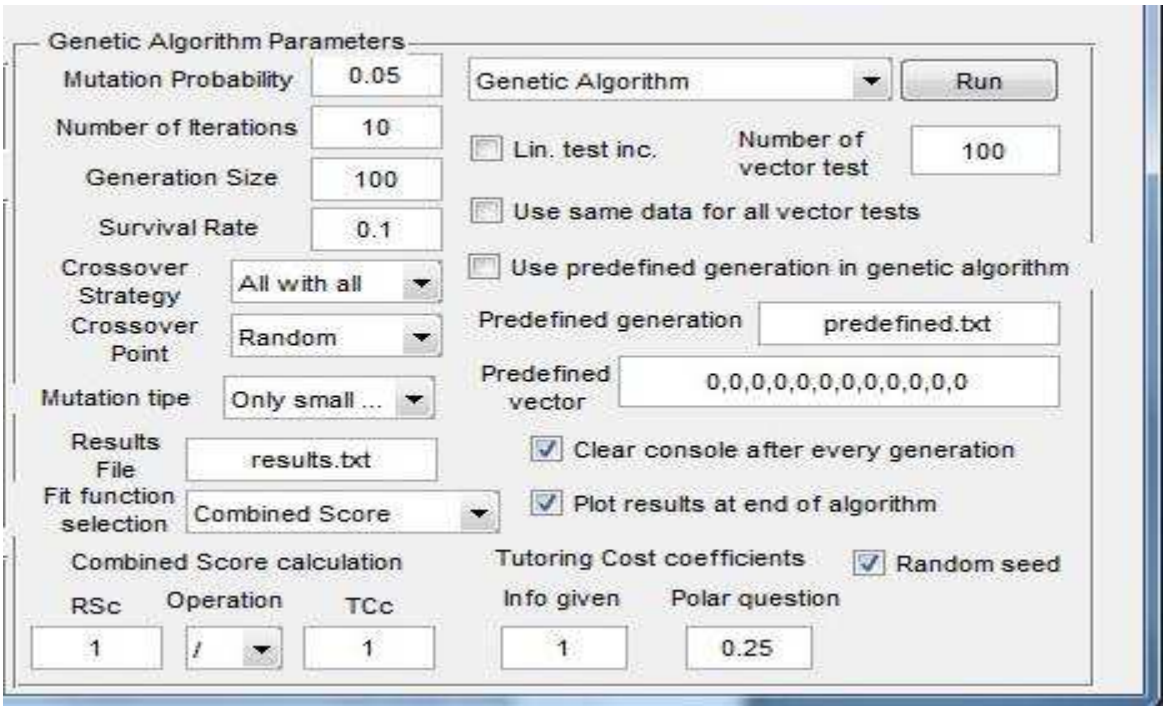
Slika 4.7: Potek genetskega algoritma za preiskovanje učnih strategij

4.5 Grafični uporabniški vmesnik

Uporabniški vmesnik se zažene ob zagonu programa. Uporabniški vmesnik je razdeljen na dva dela. Prvi, levi del služi nastavitvam ICLS-ja, torej parametrom učnega algoritma, ki ga uporabljamo. V drugem, desnem delu pa izbiramo način delovanja in potrebne parametre za preiskovanje učnih strategij. Slika 4.8 prikazuje celoten grafični uporabniški vmesnik, Slika 4.9 pa le desni del vmesnika.



Slika 4.8: Celotni grafični uporabniški vmesnik. Levi del služi za parametre ICLS-ja, desni pa za nastavitve preiskovanja učnih strategij.



Slika 4.9: Desni del grafičnega uporabniškega vmesnika, ki služi za izbiro načina delovanja in vnos parametrov za preiskovanje učnih strategij..

Program ima štiri načine delovanja. Ko izberemo način delovanja, program zaženemo s pritiskom na gumb »Run«. Način delovanja izberemo v spustnem oknu. Možni načini delovanja so sledeči:

- genetski algoritem (angl. *Genetic algorithm*): poženemo genetski algoritem za preiskovanje učnih strategij,
- testiraj predefinirano generacijo (angl. *Test predefined generation*): testira predefinirano generacijo, ki je določena v pripadajočem tekstovnem polju,
- testiraj predefiniran vektor (angl. *Test predefined vector*): testira predefiniran vektor, določen v pripadajočem tekstovnem polju,
- prikaži rezultate (angl. *Show results*): prikaže rezultate, shranjene v datoteki z rezultati, ta paje določena v pripadajočem tekstovnem polju.

Parametri gentskega algoritma, ki jih je mogoče nastaviti, so:

- verjetnost mutacije (angl. *Mutation probability*): tekstovno polje, v katerem določimo verjetnost mutacije potomcev,
- število iteracij oz. generacij (angl. *Number of iterations*): tekstovno polje, v katerem določimo, skozi koliko generacij naj genetski algoritem iterira,
- odstotek preživetja (angl. *Survival rate*): tekstovno polje, v katerem določimo, kolikšen odstotek najuspešnejših staršev uporabimo v koraku križanja,

- strategija križanja (angl. *Crossover strategy*): padajoči meni, s katerim izberemo, na kakšen način se izbereta starša za proces križanja izmed odstotka najuspešnejših staršev (določeno z odstotkom preživetja). Ima dve možnosti:
    1. vsak z vsakim (angl. *All with all*): vsak starš v skupini najuspešnejših se pokriža z vsemi ostalimi v skupini najuspešnejših,
    2. naključno (angl. *Random*): za križanje izberemo dva naključna starša izmed skupine najuspešnejših,
- točka križanja (angl. *Crossover point*): padajoči meni, s katerim določimo točko v vektorju staršev, kjer se bosta križala. Ponuja vseh 11 možnih točk in možnost naključnega izbora točke,
- tip mutacije (angl. *Mutation type*): padajoči meni, s katerim določimo vrsto mutacije. Možni sta dve izbiri:
    1. samo manjše mutacije (angl. *Only small mutations*): ta izbira omogoči le manjše spremembe v vrednostih vektorja. Npr. vrednost 1 lahko mutira le v 0 ali 2, ki sta njej najbližji vrednosti,
    2. dopusti večje mutacije (angl. *Allow big mutatinos*): ta izbira omogoča tudi večje spremembe vrednosti vektorja. Katerokoli vrednost lahko zamenja katerakoli druga možna vrednost vektorja,
- selekcija cenilne funkcije (angl. *Fit function selection*): padajoči meni, s katerim določimo, po katerem kriteriju kakovosti posameznika naj cenilna funkcija izbere najprimernejše starše v generaciji. Možne so tri izbire:
    1. uspešnost prepoznavanja (angl. *Recognition score*): cenilna funkcija bo za kriterij vzela mero prepoznavanja (angl. *Recognition score*),
    2. cena tutorstva (angl. *Tutoring cost*): cenilna funkcija bo za kriterij vzela ceno tutorstva (angl. *Tutoring cost*). Cena tutorstva je utežena vsota. Koeficiente vsote (angl. *Tutoring cost coefficient*) določimov dva tekstovna polja »Info given« (podaj informacijo) in »Polar question« (polarno vprašanje),
    3. kombinirana mera (angl. *Combined score*): cenilna funkcija bo za kriterij vzela kombinacijo med mero prepoznavanja in ceno tutorstva. Operacijo med obema merama lahko izberemo (angl. *Combined score operation*). Obe meri lahko tudi obtežimo v tekstovnih poljih RSc in TSc (angl. *Recognition score coefficient, Tutoring cost coefficient*).

Ostale nastavitve, ki jih omogoča vmesnik, so :

- število testov vektorja (angl. *Number of vector tests*): v tekstovno polje vnesemo število testiranj za vsak vektor učne strategije. Končni rezultat vektorja je povprečna vrednost vseh njegovih testov,

- linearno povečevanje testov (angl. *Linear test incrementation*): če obkljukamo potrditveno polje, se z vsako iteracijo genetskega algoritma število testov vektorja linearno poveča. Na primer, če določimo število testov vektorja 100 in imamo 10 generacij, potem se bo število testov za vektor z vsako generacijo povečalo za 10,
- uporabi iste podatke za vse vektorje (angl. *Use same data for all vector tests*): če obkljukamo potrditveno polje, za vsak test vseh vektorjev vseh generacij uporabimo iste podatke,
- uporabi predefinirano generacijo v genetskem algoritmu (angl. *Use predefined generation in genetic algorithm*): če obkljukamo potrditveno polje, omogočimo uporabo predefinirane generacije v .txt dokumentu za začetno generacijo genetskega algoritma (ne generira naključne začetne generacije). Ime datoteke lahko določimo v tekstovnem polju (angl. *Predefined generation*),
- predefinirani vektor (angl. *Predefined vector*): v to tekstovno polje lahko vnesemo poljubni učni vektor, ki ga lahko potem posamično testiramo,
- počisti prikaz ukazne vrstice po vsaki generaciji (angl. *Clear console after every generation*): če polje obkljukamo, se bo ob teku algoritma ob koncu vsake generacije konzola pobrisala,
- prikaži grafe rezultatov ob koncu algoritma (angl. *Plot results at the end of algorithm*): če obkljukamo polje, se nam ob zaključku algoritma izrišejo rezultati v obliki grafov,
- naključno seme (angl. *Random seed*): če obkljukamo polje, ob vsakem zagonu algoritma vzamemo novo seme za naključnost,
- dokument za rezultate (angl. *Results file*): v tekstovno polje vnesemo ime dokumenta, kamor želimo, da se shranijo rezultati. Dokument mora imeti končnico .txt.

4.6 Izpis poteka v ukazni vrstici

V ukazni vrstici se izpisuje potek programa. Oblika izpisa se nekoliko razlikuje glede na način delovanja, ki smo ga izbrali. Izpis poteka je prisoten pri vseh načinih, razen pri načinu za prikaz podatkov. Pri načinu za testiranje predefiniranega vektorja se izpiše tudi rezultat testiranja. Primer izpisa prikazuje Slika 4.10.

```
Genetic Algorithm parameters:
Results saved to: D:/Diploma/ICLS/results/results.txt
Predefined generation use: 0
Predefined generation location: D:/Diploma/ICLS/predefined/predefined.txt
Initial predefined generation size: 0
Random seed: 1
Number of generations (iterations): 3
Generation size: 10
Mutation probability: 0.050
Mutation tipe: Only small changes allowed
Crossover Strategy: Random
Crossover Point: Random
Survival rate for fit funcion: 0.100
Number of vector test in fit funcion: 10
Use same data for all vector test: 0
Linear test incrementation: 0
Fit Function Selection parameter: Combined Score
Tutoring cost coeficients: Info given: 1.000, Polar questions answered: 0.250
Combined Score operatin and coficients are RSk: 1.000 TCk: 1.000 OP: /

Starting fit function tests:

Testing 1 vektor of 2 generation with 10 tests: 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% Done... elapsed time is: 0 minutes an 36 seconds
Testing 2 vektor of 2 generation with 10 tests: 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% Done... elapsed time is: 0 minutes an 39 seconds
Testing 3 vektor of 2 generation with 10 tests: 10% 20% 30% 40% 50% 60% 70%>>
```

Slika 4.10: Primer izpisa poteka programa v ukazni vrstici ob zagonu genetskega algoritma.

Ob začetku vsake iteracije genetskega algoritma se izpišejo parametri, s katerimi smo ga zagnali. Nato se začne izpisovati potek testiranja cenilne funkcije za vsak vektor posebej. Izpisuje se tudi čas, ki je pretekel od začetka testiranja. Na koncu se izpiše sporočilo o zaključku algoritma. Končni izpis prikazuje Slika 4.11

```
Start test predefined vector...

Testing 1 vektor of 1 generation with 10 tests: 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% Done... elapsed time is: 0 minutes an 3 seconds
Tested vector and its results are (Recognition score, Tutoring cost, Combined score):
         0         0         0         0         0         0         0         0         0         0         0         0  119.3500    0.7000  170.5000

Elapsed time is: 0 minutes an 3 seconds
```

Slika 4.11: Primer izpisa v ukazni vrstici ob testiranju predefiniranega vektorja.

Pri testiranju predefiniranega vektorja se nam v ukazni vrstici izpisuje potek testiranja. Ob zaključku se izpišejo rezultati. Primer izpisa rezultatov prikazuje Slika 4.12.

```
Start test predefined generation...

Tested generation parameters:
Results saved to: D:/Diploma/ICLS/results/results.txt
Predefined generation location: D:/Diploma/ICLS/predefined/predefined.txt
Initial predefined generation size: 10
Survival rate for fit funcion: 0.100
Number of vector test in fit funcion: 10
Fit function selection parameter: Combined Score
Tutoring cost coeficients: Info given: 1.000, Polar questions answered: 0.250
Combined Score operatin and coficients: RSk: 1.000 TCk: 1.000 OP: /

Testing 1 vektor of 1 generation with 10 tests: 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% Done... elapsed time is: 0 minutes an 3 seconds
Testing 2 vektor of 1 generation with 10 tests: 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% Done... elapsed time is: 0 minutes an 6 seconds
Testing 3 vektor of 1 generation with 10 tests: 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% Done... elapsed time is: 0 minutes an 9 seconds
Testing 4 vektor of 1 generation with 10 tests: 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% Done... elapsed time is: 0 minutes an 12 seconds
Testing 5 vektor of 1 generation with 10 tests: 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% Done... elapsed time is: 0 minutes an 15 seconds
Testing 6 vektor of 1 generation with 10 tests: 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% Done... elapsed time is: 0 minutes an 18 seconds
Testing 7 vektor of 1 generation with 10 tests: 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% Done... elapsed time is: 0 minutes an 21 seconds
Testing 8 vektor of 1 generation with 10 tests: 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% Done... elapsed time is: 0 minutes an 25 seconds
Testing 9 vektor of 1 generation with 10 tests: 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% Done... elapsed time is: 0 minutes an 28 seconds
Testing 10 vektor of 1 generation with 10 tests: 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% Done... elapsed time is: 0 minutes an 31 seconds

Test predefined generation results are saved to D:/Diploma/ICLS/results/results.txt

Elapsed time is: 0 minutes an 31 seconds

...End predefined generation test
```

Slika 4.12: Primer izpisa v ukazni vrstici ob testiranju predefinirane generacije.

Ko se testira predifinirana generacija, je izpis v ukazni vrstici o poteku zelo podoben izpisu ob poteku genetskega algoritma. Najprejse izpišejo parametri, ki se uporabljajo ob testiranju,nato pa se sproti izpisuje potek testiranja vsakega vektorja v generaciji.

4.7 Shranjevanje rezultatov

Rezultati genetskega algoritma in testiranja predefinirane generacije se shranjujejo v preprosto tekstovno datoteko. Formata, v katerega se rezultati shranijo, sta v obeh primerih podobna. V glavi datoteke se najprej zapišejo uporabljeni parametri pri zagonu algoritma, pri testiranju predefinirane generacije pa se nato rezultati shranijo ob koncu testiranja. Shranijo se testirani vektorji v generaciji in vrednosti mer za ocenjevanje njihove kakovosti. V primeru genetskega algoritma se najboljši vektorji shranijo ob koncu testa vsake generacije. Na koncu se še enkrat shranijo najboljši vektorji vseh generacij in vrednosti njihovih mer kakovosti, kar se uporablja za prikazovanje rezultatov v obliki grafov. Primer datoteke z rezultati prikazuje Slika 4.13.

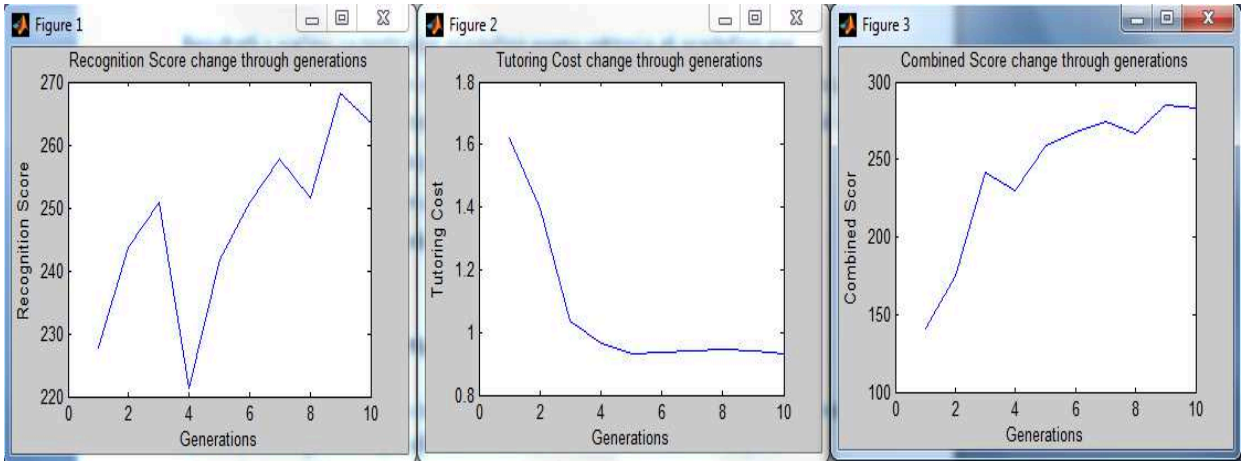
```
1
2   Genetic Algorithm parameters:
3   Results saved to: D:/Diploma/ICLS/results/results.txt
4   Predefined generation use: 0
5   Predefined generation location: D:/Diploma/ICLS/predefined/predefined.txt
6   Initial predefined generation size: 0
7   Random seed: 1
8   Number of generations (iterations): 3
9   Generation size: 10
10  Mutation probability: 0.050
11  Mutation tipe: Only small changes allowed
12  Crossover Strategy: All with all
13  Crossover Point: Random
14  Survival rate for fit funcion: 0.500
15  Number of vector test in fit funcion: 1
16  Use same data for all vector test: 0
17  Linear test incrementation: 0
18  Fit Function Selection parameter: Combined Score
19  Tutoring cost coeficients: Info given: 1.000, Polar questions answered: 0.250
20  Combined Score operatin and coficients are RSk: 1.000 TCk: 1.000 OP: /
21
22  Genetic algorithm results:
23
24  Best of iteration 1, elapsed time 0 minutes and 5 seconds
25
26  0,0,0,2,1,-3,-1,-2,-1,-3,-3,-2,286.5,1.375,208.36
27  1,2,3,0,1,2,1,3,0,0,-2,2,204.5,1.455,140.55
28  -2,2,2,-3,3,1,-3,-2,2,-1,-1,3,139,1.875,74.133
29  -3,-2,-1,-1,-3,-1,-2,-1,1,-3,-2,3,136.5,2.57,53.113
30  1,1,-1,-3,0,-1,-3,0,-3,-2,-1,2,48,1.085,44.24
31
32  Best of iteration 2, elapsed time 0 minutes and 9 seconds
33
34  1,2,3,0,1,2,1,3,0,0,-1,3,220,0.845,260.36
35  0,0,0,2,1,1,-3,-2,2,-1,-1,3,181,0.885,204.52
36  0,2,3,0,1,2,1,3,0,0,-2,2,200,0.995,201.01
37  0,0,0,2,1,-3,-2,-2,-1,-3,-3,-2,240,1.32,181.82
38  1,2,3,0,1,2,1,3,0,0,-2,2,213,1.21,176.03
39
40  Best of iteration 3, elapsed time 0 minutes and 13 seconds
41
42  1,2,3,0,1,2,1,3,0,0,-1,3,281.5,0.81,347.53
43  0,0,0,2,1,1,-3,-2,2,-1,-1,2,224,0.745,300.67
44  0,0,-1,2,1,1,-3,-2,2,-1,-1,3,229.5,0.85,270
45  0,0,0,2,1,1,-3,-2,0,0,-1,3,223.5,0.87,256.9
46  1,2,0,2,1,-3,-2,-2,-1,-3,-3,-2,288,1.345,214.13
47
48  The best vector of each iteration:
49
50  0,0,0,2,1,-3,-1,-2,-1,-3,-3,-2,286.5,1.375,208.36
```

Slika 4.13: Primer datoteke z rezultati za genetski algoritem.

Ime datoteke za rezultate vnesemo v namensko tekstovno polje »Results file« v grafičnem vmesniku. Datoteka mora imeti končnico .txt in se shrani v mapo »results«.

4.8 Prikaz rezultatov

Rezultati v načinu za testiranje predefiniranega vektorja ali predefinirane generacije vektorjev se nam ob koncu testiranj izpišejo v ukazni vrstici. Rezultate genetskega algoritma, shranjene v datoteki, lahko prikažemo v obliki grafov. To lahko storimo tako, da obkljukamo opcijo prikaza rezultatov ob koncu algoritma v grafičnem uporabniškem vmesniku. Kadarkoli pa lahko shranjene rezultatev datoteki spet prikažemo, če izberemo način za prikaz rezultatov. Primer prikaza rezultatov vidimo na Sliki 4.14.



Slika 4.14: Primer izpisa rezultatov v obliki grafov.

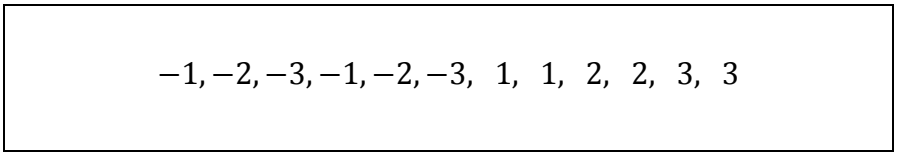
Prikažejo se nam trije grafi, na katerih vidimo spreminjanje mere prepoznavanja, cene tutorstva in kombinirane mere skozi generacije. Za koordinato mer kakovosti se vzame vrednost mere najboljšega vektorja v generaciji.

4.9 Uporaba programa

Program začnemo tako, da zaženemo datoteko LLS.m. V grafični vmesnik vnesemo želene parametre in način delovanja ter pritisnemo na gumb »Run«. Delovanje programa lahko spremljamo v ukazni vrstici, kjer se nam izpiše tudi opozorilo o zaključku. Nato lahko prek grafičnega vmesnika spet zaženemo želeni način.

Predefinirani vektor

V namensko tekstovno polje vnesemo vrednosti vektorja, ki ga želimo testirati. Vrednosti vektorja so lahko cela števila od –3 do 3. Vrednosti ločimo z vejico. Vektor je zapisan na isti način, kot se uporablja vektorski zapis učne strategije v celotnem programu. Primer vektorja prikazuje Slika 4.15.

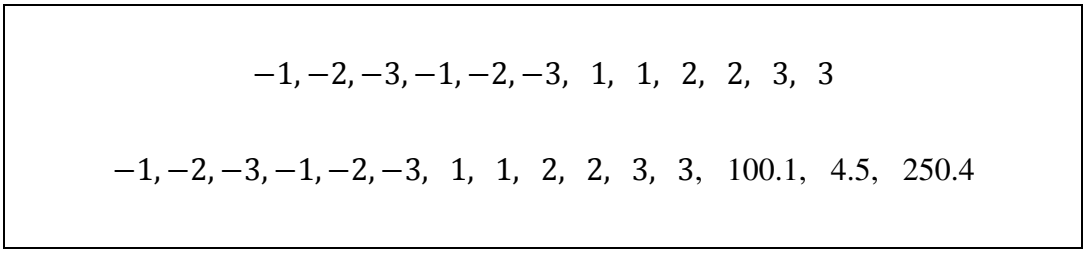


Slika 4.15: Primer vektorja, ki ga vnesemo v namensko tekstovno polje

Po vnosu vektorja izberemo način za testiranje predefiniranega vektorja in poženemo program.

Predefinirana generacija

Predefinirano generacijo določimo v datoteki. Ime datoteke vnesemo v namensko polje in mora imeti končnico txt. Datoteka se nahaja v mapi »predefined«. Vektorje v datoteki s predefinirano generacijo je treba vpisati v določenem formatu. Vsaka vrstica je en vektor. Vektor mora biti vnešen v obliki, ki se uporablja v celotnem programu. Primer oblike vektorja je na Sliki 4.16.



Slika 4.16: Primer dveh oblik vektorja v predefinirani generaciji.

Poleg vrednosti vektorja lahko dodamo še tri vrednosti mer za ta vektor. To omogoči, da lahko enostavno kopiramo generacijo iz datoteke z rezultati. Namen tega je, da lahko v primeru prekinitve programa med izvajanjem genetskega algoritma ponovno poženemo algoritem od zadnje pretestirane generacije s pomočjo predefinirane generacije. V temprimeru moramo obkljukati polje za uporabo predefinirane generacije in zagnati genetski algoritem. Predefinirano generacijo pa lahko testiramo samo zase, če izberemo tak način delovanja. Primer predefinirane generacije vidimo na Sliki 4.16.

```
1   3,3,-2,-1,1,-1,-2,1,0,-1,-1,3,219.5,1.07,205.14
2   3,1,3,1,3,3,2,-3,-2,-3,-3,1,213.5,1.7,125.59
3   1,3,-1,-1,0,2,3,-3,-3,-3,1,2,105,1.28,82.031
4   -1,-3,-3,2,1,2,1,2,0,-2,-2,3,209.5,3.085,67.909
5   2,0,3,2,-3,-2,-1,3,0,0,1,-2,77,1.585,48.58
6   -2,2,1,-2,-3,-3,0,-1,0,-2,3,0,125.5,3.065,40.946
7   1,2,-2,1,0,2,-2,-1,3,-2,3,2,90.5,3.08,29.383
8   -3,3,2,3,-2,-2,1,0,1,2,-1,3,-44.5,1.43,-31.119
9   2,-1,0,-3,2,3,3,2,-1,1,0,0,-93,2.15,-43.256
10  -2,-3,0,-2,-3,-1,1,3,2,3,2,1,-134,1.755,-76.353
11
```

Slika 4.16: Primer .txt datoteke, ki vsebuje predefinirano generacijo.

Prikaz rezultatov

Rezultate lahko ob koncu genetskega algoritma prikažemo, če obkljukamo namensko polje v grafičnem vmesniku. Rezultate si lahko po shranitvi kadarkoli pogledamo tako, da v namensko polje določimo ime datoteke z rezultati, ki jih želimo pokazati, izberemo način za prikaz rezultatov in poženemo program.

Genetski algoritem

Za zagon genetskega algoritma izberemo ta način in poženemo program. Določimo tudi parametre za njegovo delovanje. Pole običajnih parametrov genetskega algoritma, ki jih določimo v grafičnem vmesniku, lahko določimo še nekaj dodatnih možnosti. V grafičnem vmesniku lahko izberemo:

- linearno povečevanje testov. Če želimo nekoliko pohitriti genetski algoritem, lahko obkljukamo to polje. S tem bo program v zgodnjih generacijah vektorje testiral manjkrat in število testov povečeval linearno z vsako sledečo generacijo. V prvih generacijah tako vektorje bolj na grobo prečistimo v slabše in boljše. Kasneje, ko so razlike v kakovosti manjše, saj smo najslabše vektorje prečistili že prej, pa z več testi dobimo natančnejšo oceno,
- uporabi iste podatke za vse teste vektorjev. Smiselno je uporabiti ob 1. testu za vsak vektor. Če polje ni obkljukano, se za vsak test uporabijo drugi učni primeri, in je končni rezultati povprečna vrednost posameznih testov,
- uporabi predefinirano generacijo,
- počisti ukazno vrstico ob koncu generacije. V določenih primerih naredi spremljanje poteka genetskega algoritma prek ukazne vrstice prijaznejše,
- izriši grafe rezultatov na koncu algoritma,
- naključno seme.

5 Meritve in rezultati

Pri iskanju najboljše učne strategije oziroma optimiziranju učnih strategij se seveda najprej postavi vprašanje, kaj je to dobra strategija. Nekako moramo znati ovrednotiti kakovost učnih strategij, da lahko te med seboj primerjamo. Kot smo že večkrat omenili, nam kakovost učne strategije poda cenilna funkcija genetskega algoritma, ta je pridobljena prek interativnega učnega algoritma, ki naučeno znanje testira na primerih. Funkcija nam kot rezultat kakovosti učnega algoritma vrne dve meri. Prva je uspešnost prepoznavanja, ki nam pove, koliko od testnih primerov, naučenih konceptov, je znal učni algoritem po učenju pravilno prepoznati. Poda nam oceno, kako dobro je naučeno znanje. Končni cilj vsakega učenja je, da je naučeno znanje čimbolj kakovostno. Pri samem učnem procesu interaktivnega učenja pa je treba upoštevati še en faktor, in sicer tutorja. Dobra učna strategija bi bila takšna, s katero se učenec dobro nauči nečesa, in hkrati tudi takšna, ki od mentorja terja čim manj posredovanja oziroma ukvarjanja z učencem. Lahko bi sklepali, da načeloma za boljše znanje učenca potrebujemo več dela s strani tutorja. Dobra učna strategija bi bila lahko takšna, ki ob čim manjšem delu tutorja čimveč nauči učenca. Glede na mere, s katerimi merimo kakovost učne strategije, to pomeni, da iščemo tako strategijo, ki ima čim večjo mero prepoznavanja in hkrati čim manjšo možno ceno tutorstva.

5.1 Rezultati

Za meritve smo uporabili naslednji nabor parametrov za genetski algoritem:

- verjetnost mutacije: 0,05,
- število iteracij: 10,
- velikost generacije: 100,
- odstotek preživetja: 10%,
- strategijo križanja: naključno,
- točko križanja: naključno,
- tip mutacije: dovoli velike spremembe.

Za učni algoritem smo uporabili Gaussian of Best Feature, parametri tega pa so bili sledeči:

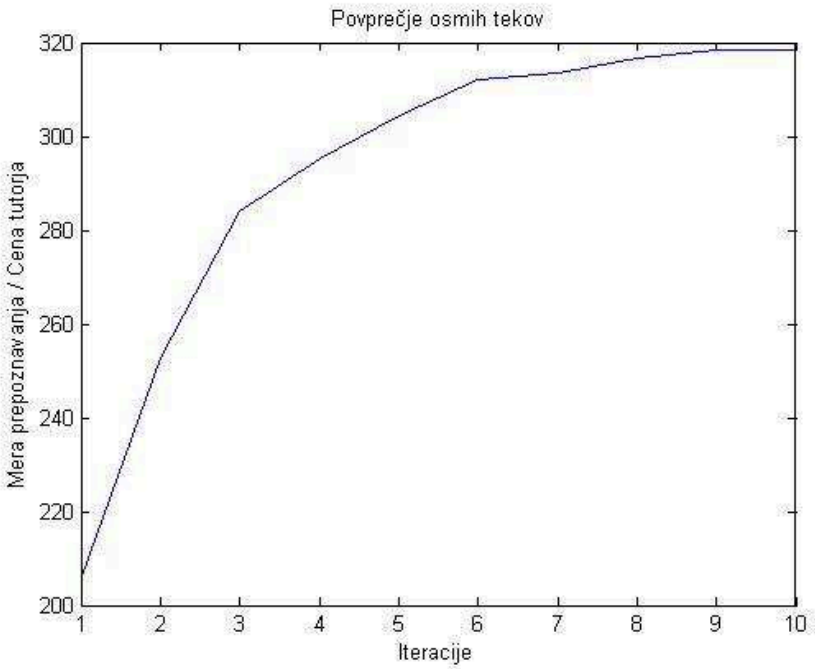
- število slik: 50,
- število testnih slik: 50,
- število slik, ki se bodo obdelale z začetno učno strategijo: 20,
- po katerih učnih slikah naj se izvede test: 50.

Vsakič, ko se je učni algoritem zagnal v cenilni funkciji, je za zgornje parametre naključno izbral slike iz množice 500 slik. Algoritem ne deluje neposredno s slikami, ampak uporablja šest značilnic. Prve tri so mediana HSV (hue, saturation, value)in tri značilnice oblik. Učni algoritem se v našem primeru uči, katere barve je predmet: rdeče, zelene, modre ali rumene. Primer slik vidimo na Sliki 5.1 [5].

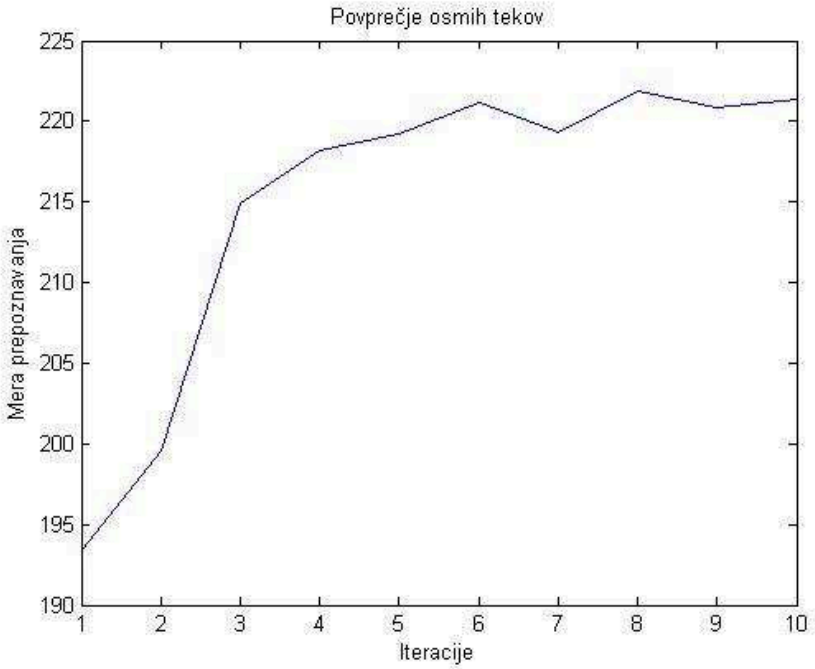
Slika 5.1. Primeri uporabljenih slik.

5.1.1 Razmerje med uspešnostjo prepoznavanja in ceno tutorstva

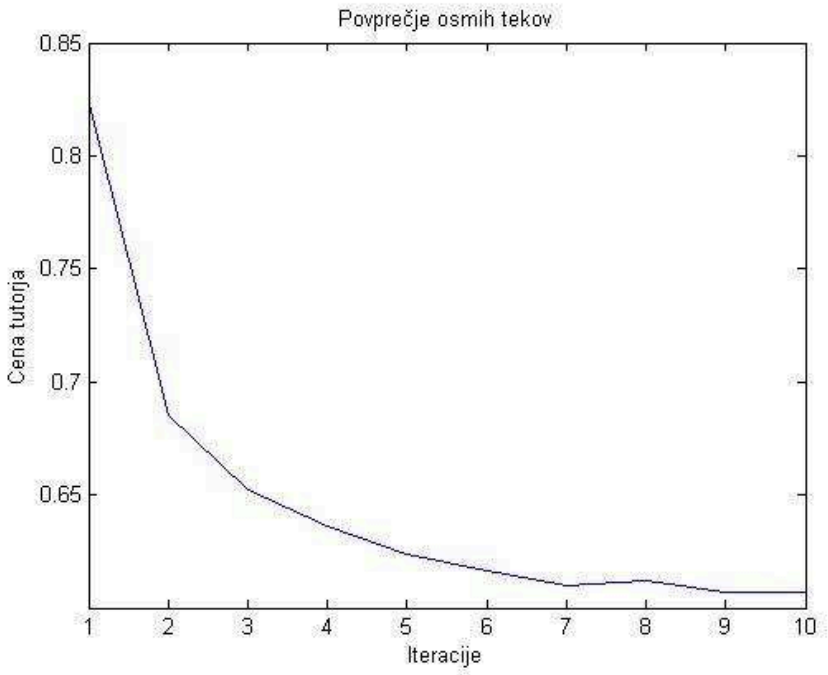
Vsak vektor je cenilna funkcija testirala stokrat in rezultat povprečila. Za merilo uspešnosti smo uporabili razmerje med uspešnostjo prepoznavanja in ceno tutorstva. Zaradi tega smo pričakovali, da bo ta mera skozi generacije naraščala, kar pomeni, da bi se uspešnost prepoznavanja večala in cena tutorstva manjšala z vsako generacijo. Algoritem smo s temi parametri pognali osemkrat in rezultate povprečili. Grafi 5.1, 5.2 in 5.3 prikazujejo povprečene vrednosti najboljših vektorjev izmed vsake generacije za osem tekov algoritma. Točko na grafu predstavlja vrednost najboljšega vektorja generacije.



Slika 5.1: Graf spreminjanja razmerja med uspešnostjo prepoznavanja in ceno tutorstva po iteracijah.



Slika 5.2: Graf spreminjanja uspešnosti prepoznavanja po iteracijah.



Slika 5.3: Graf spreminjanja cene tutorja po iteracijah.

Iz vseh treh grafov vidimo, da se je učinkovitost učnih strategij skozi generacije izboljšala. Uspešnost prepoznavanja je naglo narasla prvih nekaj generacij, nato pa se je njeno izboljšanje spreminjalo bolj umirjeno. Podobno velja za ceno tutorstva,kise je zniževala, kar pomeni, da je imel tutor pri učni strategiji z vsako iteracijo manj dela. Zviševanje uspešnosti prepoznavanja in zmanjševanje cene tutorstva privede do izboljšave učnih strategij v naslednji generaciji. Njuno razmerje se skladno z njima hitreje povečuje prvih nekaj generacij, nato pa počasneje izboljšuje. Poglejmo še, kakšne vektorje smo dobili na koncu osmih tekov. Z zeleno barvo so označene vrednosti, pri katerihje zlato pravilo pozitivno, pri rdečih pa negativno. Na Sliki 5.4 je prikazanih 10 najboljših vektorjev zadnje generacije.

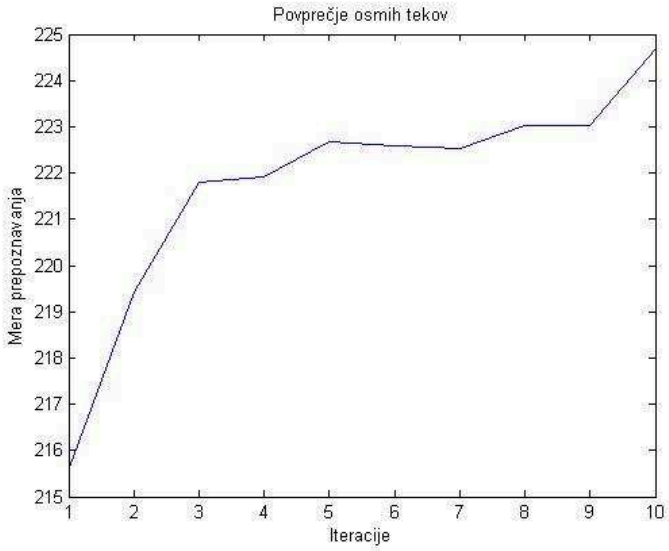
[0,0,3,1,1,2,0,-1,0,-1,0,-1]
[0,-1,3,1,1,1,-1,0,0,0,-1,-2]
[-1,1,0,1,1,2,-1,-1,0,0,-1,2]
[1,-1,-1,1,1,1,0,-1,0,0,0,2]
[0,-1,0,1,1,3,0,0,0,0,0,2]
[0,1,-1,1,1,0,-1,0,0,-1,-1,-1]
[-1,0,1,1,1,-1,0,-1,0,0,-1,3]
[1,0,1,1,1,3,-1,0,-1,0,-1,3]

Slika 5.4: Najboljših 10 vektorjev zadnje generacije

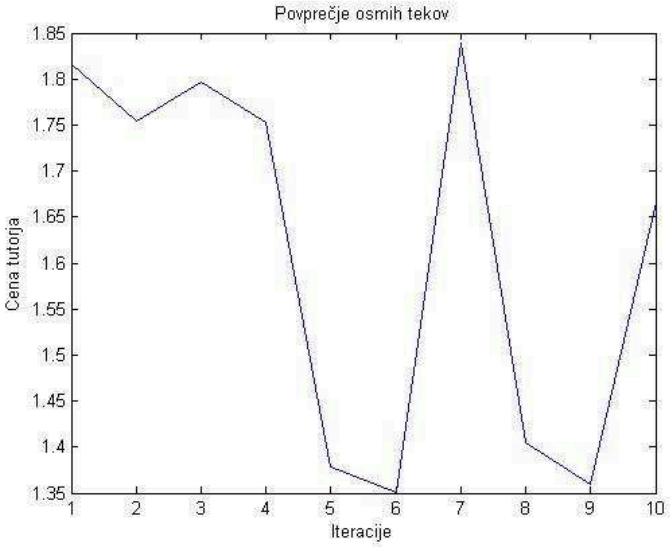
Vidimo, da se največkrat pojavita akcija samodejne popravitve znanja (angl. *update*), akcija ne naredi ničesar (angl. *do nothing*). Posodabljanje znanja je v večini primerov opravljeno z negativnim primerom, kadar je bilo prepoznavanje koncepta napačno glede na zlato pravilo, to pravilo s pozitivnim primerom pa takrat, kadar je bilo prepoznavanje pravilno. Ostale akcije se pojavijo relativno redko. Iz grafa za ceno tutorstva vidimo, da je ta pri končnih strategijah relativno nizka, kar se odraža tudi v vektorjih, saj samodejno posodabljanje ne terja dela tutorja, ravno tako pa akcija ne naredi ničesar.

5.1.2 Uspešnost prepoznavanja

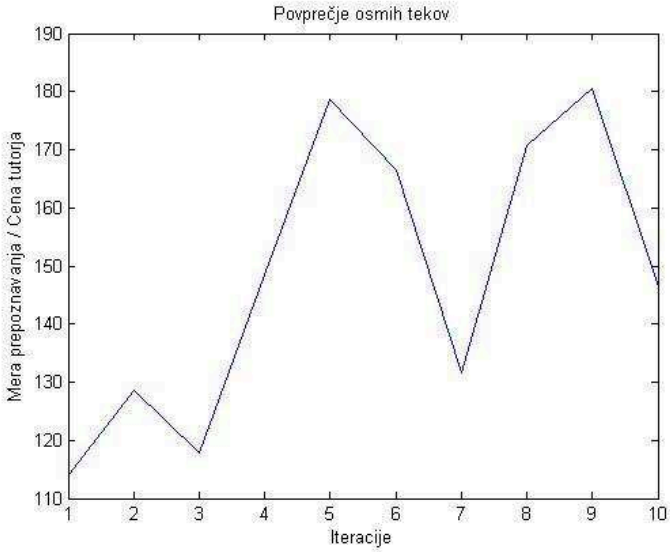
Zanimalo nas je še, kaj se zgodi, če za merilo uspešnosti uporabimo le uspešnost prepoznavanja ali samo ceno tutorstva. Algoritem smo pognali z istimi parametri genetskega algoritma, le merilo uspešnosti je bilo drugačno. Najprejsi poglejmo rezultate z uspešnostjo prepoznavanja za merilo uspešnosti, ki jih vidimo na slikah 5.5, 5.6 in 5.7.



Slika 5.5: Graf spreminjanja uspešnosti prepoznavanja po iteracijah.



Slika 5.6: Graf spreminjanja cene tutorja po iteracijah.



Slika 5.7: Graf spreminjanja razmerja med uspešnostjo prepoznavanja in ceno tutorstva po iteracijah.

Pričakovano se uspešnost prepoznavanja lepo izboljšuje skozi generacije, medtem pa, ker sta cena tutorstva in razmerje med obema merama nepomembni pri oceni, popolnoma podivjata in močno nihata. Vektorji, ki smo jih dobili pri teh meritvah, so na Sliki 5.8.

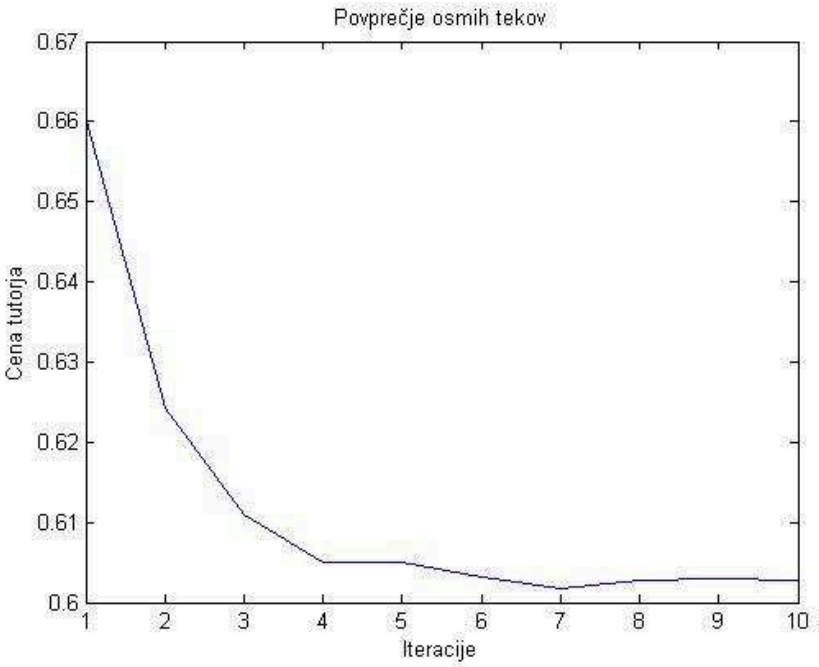
[-1,1,3,1,3,2,0,0,-3,0,-1,-3]
[-1,0,1,3,1,1,-1,-3,-2,-3,-3,0]
[-2,1,3,2,1,1,-3,-1,0,-2,-1,2]
[-3,1,3,3,2,-3,-2,0,0,-3,-1,-2]
[-1,3,2,1,1,2,-1,0,-2,-2,-1,-2]
[-3,1,3,1,2,3,-2,0,-2,0,-3,-1]
[-2,-3,-2,2,2,3,-3,-2,-1,-1,0,-3]
[-2,0,3,1,1,-2,-3,0,-2,0,-2,1]

Slika 5.8: Najboljših 10 vektorjev zadnje generacije glede na uspešnost prepoznavanja.

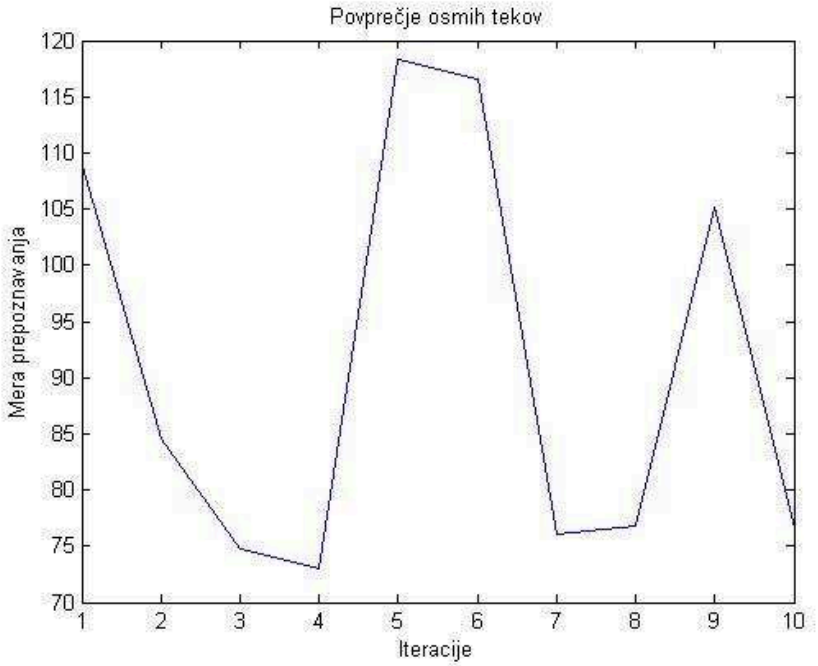
S prejšnjimi vektorji imajo ti skupnega to, da uporabljajo večinoma pozitivne ali negativne primere za posodabljanje znanja v istih situacijah. Hitro lahko opazimo, da so akcije precej bolj raznolike, število posameznih akcij je približno enako, kar kaže na to, da strategije pri iskanju dobre mere prepoznavanja niso skrbele istočasno za nizko delo tutorja.

5.1.3 Cena tutorstva

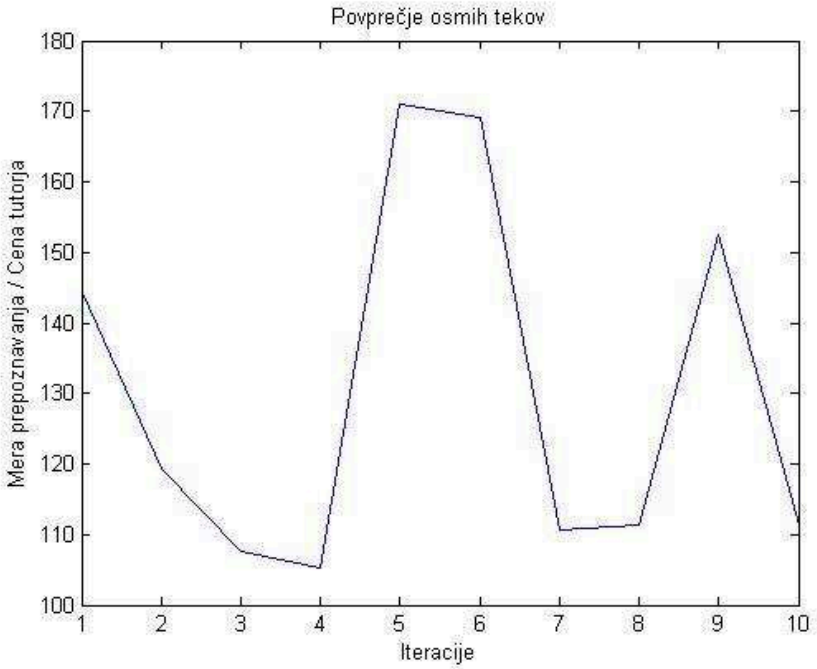
Na zadnje poglejmo še rezultate, če je ocena kakovostiučne strategije odvisna le od cene tutorstva. Ti rezultati so vidni na Slikah 5.9, 5.10 in 5.11.



Slika 5.9: Graf spreminjanja razmerja med uspešnostjo prepoznavanja in ceno tutorstva po iteracijah.



Slika 5.10: Graf spreminjanja uspešnosti prepoznavanja po iteracijah.



Slika 5.11: Graf spreminjanja razmerja med uspešnostjo prepoznavanja in ceno tutorstva po iteracijah.

Genetski algoritem v teh primerih pravilno čimbolj zmanjša ceno tutorstva, ostali dve meri pa nihata nenadzirano, saj nista upoštevani v oceni kakovosticenilne funkcije algoritma. Pripadajoči vektorji tem rezultatom so na Sliki 5.12.

[1,1,-1,-1,-1,2,0,1,1,-1,1,3]
[-1,0,1,1,1,1,0,0,0,1,1,-3]
[-1,0,-1,0,-1,3,1,0,1,-1,1,1]
[-1,-1,0,1,1,1,1,-1,-1,-1,1,3]
[1,-1,1,0,0,-1,1,0,1,-1,0,2]
[0,1,0,1,0,-1,0,0,1,0,0,-3]
[1,1,0,0,-1,2,1,1,0,0,0,-2]
[0,1,-1,1,-1,-1,0,-1,1,-1,0,2]

Slika 5.12: Najboljših 10 vektorjev zadnje generacije glede na ceno tutorstva.

Pričakovano vidimo, da močno prevladujejo akcije, ki ne vpletajo tutorja. To so samodejna posodobitev znanja, ki ne naredi ničesar. Pri teh rezultatih so druge akcije, ki so tutorsko bolj drage, najmanj prisotne glede na ostale rezultate.

5.1.4 Spreminjanje vektorja skozi čas

Zanimivo je pogledati, kako se najboljši vektor vsake generacije spreminja skozi čas. Na Sliki 5.13 vidimo spremembo najboljšega vektorja skozi generacije pri izvajanju poskusa. Pri tem poskusu je bilo za merilo uspešnosti uporabljeno razmerje med uspešnostjo prepoznavanja in ceno tutorstva.

[-3,0,1,1,2,1,3,-3,0,-1,-3,2]
[0,2,3,0,1,-2,-1,-1,1,-1,0,-1]
[0,2,3,3,1,-2,0,-1,-3,-1,0,-1]
[0,1,3,3,1,-2,0,-1,-3,-1,0,-1]
[0,1,3,3,1,-2,0,-1,0,-1,0,2]
[0,1,3,3,1,-2,0,-1,-3,-1,0,-1]
[0,0,3,1,1,-2,0,-1,0,-1,0,2]
[0,-1,3,1,1,1,0,-1,-3,-1,0,-1]
[0,0,3,1,1,-2,0,-1,0,-1,0,2]
[0,0,3,1,1,2,0,-1,0,-1,0,-1]

Slika 5.13: Spreminjanje najboljšega vektorja skozi generacije.

Vektor, ki je na sliki čisto zgoraj, je najboljši vektor prve generacije, vektor, ki je na sliki čisto spodaj, pa je najboljši vektor zadnje generacije. Opazimo lahko, da najboljši vektorji skozi vsako generacijo prejmejo eno ali dve spremembi. Največja sprememba je iz prve v drugo generacijo, kar lahko razložimo s tem, da so v prvi generacije popolnoma naključni vektorji, v drugi generaciji pa so že nekoliko usmerjeni skozi evolucijski proces. Zato so si vektorji v sledečih generacijah bolj podobni. Vidimo, da se na levi polovici vektorjev ohranjajo večinoma akcije s pozitivnimi primeri učenja, tiste z negativnimi pa se izločijo. Na primer, v sedmi generaciji se pojavi v drugi dimenziji vektorja negativni znak, ki se v naslednji generaciji popravi na pozitivnega. V zadnjem vektorju so tudi vse dimenzije leve polovice vektorja označene s pozitivnim predznakom. Obratno na desni polovici vektorjev vidimo, da se tam pretežno gibljejo oznake z negativnim predznakom, kar označuje učenje z negativnim primerom. Morebitne nove pozitivne vrednosti se v levi polovici izločajo in skozi generacije počasi preidejo v negativne.

5.2 Časovna zahtevnost

Poleg merjenja uspešnosti prepoznavanja in cene tutorstva program za preiskovanje učnih strategij meri tudi čas izvajanja. Po vsaki generaciji shrani pretečen čas. Če pogledamo čas za izvedbo ene generacije pri zgornjih meritvah, vidimo, da se je ena generacija izvajala približno 75 minut. Vsaka generacija vsebuje 100 vektorjev, za vsakega od njih pa cenilna funkcija ponovi testiranje na naključnih slikah stokrat in rezultat povpreči. To pomeni, da je čas enega testiranja približno 4,5 sekunde. Posamezno testiranje zavzema začetno učenje na 20 slikah, nato posodobitveno učenje na 50 slikah in na koncu preizkušanje pridobljenega znanja na nadaljnjih 50 slikah. Ker smo algoritem pustili teči 10 generacij, je trajalo celotno izvajanje približno 750 minut, kar znaša dvanajst ur in pol.Vsako testiranje smo ponovili osemkrat,kar pomeni, da smo za povprečene rezultate osmih testiranj potrebovali približno 100 ur. Pognali smo tri sklope testiranj z različnimi cenilnimi funkcijami, kar pomeni, da smo za vse zgoraj dobljene rezultate porabili 300 ur.

6 ZAKLJUČEK

Za diplomsko delo smo si zadali cilj najti čim bolj učinkovito učno strategijo za interaktivno učenje. Najprej smo predstavili interaktivno učenje kot učenje učenca s pomočjo tutorja. Način komunikacije med tema akterjema definira učna strategija. Predstavili smo tudi formalizacijo učnih strategij in štiri osnovne učne strategije. Za iskanje novih učnih strategij smo izbrali genetski algoritem. Te algoritme smo na splošno opisali in predstavili idejo, kako bi jih uporabili za iskanje čim boljše učne strategije. Genetski algoritem smo implementirali v program. Opisali smo naš program in program ICLS ter prikazali, kako ju uporabljamo. Za merjenje uspešnosti učnih strategij smo uporabili dve meri, to sta uspešnost prepoznavanja naučenih konceptov in cena tutorstva, ki nam podaja, koliko dela je imel tutor z učencem medučnim procesom. Iskali smo tako učno strategijo, ki ima čimvečjo uspešnost prepoznavanja ob čim manjšem delu tutorja, torej smo morali razmerje med tema dvema maksimizirati. To razmerje smo uporabili za vrednotenje v cenilni funkciji genetskega algoritma. Rezultati so bili pričakovani, saj je algoritem pravilno poskušal iskati učne strategije, ki so bile v skladu s to mero čim boljše. Za cenilno funkcijo smo nato izbrali še uspešnost prepoznavanja in ceno tutorstva, njihovo razmerje nas takrat ni zanimalo. Tudi pri teh testiranjih se je algoritem obnašal pravilno inposkušal maksimizirati eno od teh mer ne glede na drugo. Na koncu poglavja je še predstavljeno časovno trajanje celotnega testiranja. V zaključku smo povzeli celotno delo.

Za nadaljnje delo bi bilo zanimivo uporabiti učni algoritem, ki omogoča izbris oz. popravljanje napačno pridobljenega znanja (angl. *unlearning*). Z upoštevanjem slednjega bi bile dobljene učne strategije bližje realnemu obnašanju med akterjema učitelja in učenca. Za dodatno obširnejše testiranje je tudi nujna pohitritev programa. Testiranje posamezne generacije s cenilno funkcijo v genetskem algoritmu se lahko izvaja paralelno, kar bi lahko bistveno pohitrilo delovanje programa. Naš algoritem strmi k temu, da maksimizira razmerje med uspešnostjo prepoznavanja in ceno tutorstva, lahko pa bi namesto tega uporabili drugačen algoritem za doseganje tega cilja.

Literatura in viri

[1] Pea, R.D. (1993). »*Distributed cognitions: Psychological and educational considerations,*«. Poglavje Practices of distributed intelligence and designs for education, 47-87. Cambridge University Press, New York.

[2] Vygotsky, L. (1962). »*Thought and Language,*«Cambridge, MA: MIT Press.

[3] Weng, J., McClelland, J., Pentland, A., Sporns, O., Stockman, I., Sur, M., Thelen, E. (2001), »Autonomus mental development by robots and animals«, *Science*, 291(5504):599 – 600.

[4] Thomaz, A. L. (2006). »*Socially Guided Machine Learning*,« Dokorska naloga, Massachusetts Institute of Technology.

[5] Danijel Skočaj, Matej Kristan, Aleš Leonardis (2009). »*Formalization of different lerning strategies in a continuous learning framework*«, proceedings of the Ninth International Conference on Epigenetic Robotics, Venice, Italy,153-160.

[6] Charles Darwin (1859). »*On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*«. London: John Murray. 1. izdaja.

[7] Andrej Dobnikar, Branko Štern (2008). »*Mehko računanje za modeliranje, razpoznavanje in regresijo*«, Fakulteta za računalništvo in informatiko, 1. Izdaja.

[8] John Henry Holland (1975). »*Adaptation in Natural and Artificial Systems«,* University of Michigan Press.

[9] Danijel Skočaj, Barry Ridge, Aleš Leonardis (2006). »On different models of continuous learning of visual properties«, Zbornik petnajste mednarodne Elektrotehniške in računalniške konference ERK, Portorož, Slovenija,105-108.